Allen-Bradley

Logix5000™ Controllers Common Procedures

1756-Lx, 1769-Lx, 1789-Lx, 1794-Lx

Programming Manual

Rockwell Automation

Important User Information

Because of the variety of uses for the products described in this publication, those responsible for the application and use of this control equipment must satisfy themselves that all necessary steps have been taken to assure that each application and use meets all performance and safety requirements, including any applicable laws, regulations, codes and standards.

The illustrations, charts, sample programs and layout examples shown in this guide are intended solely for purposes of example. Since there are many variables and requirements associated with any particular installation, Allen-Bradley does not assume responsibility or liability (to include intellectual property liability) for actual use based upon the examples shown in this publication.

Allen-Bradley publication SGI-1.1, *Safety Guidelines for the Application, Installation and Maintenance of Solid-State Control* (available from your local Allen-Bradley office), describes some important differences between solid-state equipment and electromechanical devices that should be taken into consideration when applying products such as those described in this publication.

Reproduction of the contents of this copyrighted publication, in whole or part, without written permission of Rockwell Automation, is prohibited.

Throughout this manual we use notes to make you aware of safety considerations:

ATTENTION

Identifies information about practices or circumstances that can lead to personal injury or death, property damage or economic loss

Attention statements help you to:

- identify a hazard
- avoid a hazard
- recognize the consequences

IMPORTANT

Identifies information that is critical for successful application and understanding of the product.

Allen-Bradley, ControlLogix, DH+, Logix5000, PLC-5, RSLogix 5000, RSLinx, RSNetWorx, and SLC are trademarks of Rockwell Automation.

ControlNet is a trademark of ControlNet International, Ltd.

Ethernet is a trademark of Digital Equipment Corporation, Intel, and Xerox Corporation.

Introduction

This release of this document contains new and updated information. To find new and updated information, look for change bars, as shown next to this paragraph.

Updated Information

The document contains the following changes:

This new or updated information:	Starts on page:
Plan Your Tags	3-1
Get and Set System Data	5-4
Communicate with an ASCII Device	12-1
Process ASCII Characters	13-1
Monitor Minor Faults	17-1
Store and Load a Project Using Nonvolatile Memory	19-1
Major Fault Codes	A-1
Minor Fault Codes	A-3
IEC61131-3 Compliance	B-1
Glossary terms	Glossary
ASCII Character Codes	back cover

Notes:

Purpose of this Manual

This manual guides the development of projects for Logix5000 controllers. It provides step-by-step procedures on how to perform the following tasks, which are common to all Logix5000 controllers:

- Manage Project Files
- Organize Your Logic
- Organize Tags
- Program Routines
- Test a Project
- Handle Faults

The term *Logix5000 controller* refers to any controller that is based on the Logix operating system, such as:

- CompactLogix™ controllers
- ControlLogix™ controllers
- FlexLogix™ controllers
- SoftLogix™ controllers

This manual works together with user manuals for your specific type of controller. The user manuals cover tasks such as:

- Place and configure I/O
- Communicate with devices over various networks
- Maintain the battery

Who Should Use this Manual

This manual is intended for those individuals who program applications that use Logix5000 controllers, such as:

- software engineers
- control engineers
- application engineers
- instrumentation technicians

When to Use this Manual

Use this manual when you perform these actions:

- develop the basic code for your application
- modify an existing application
- perform isolated tests of your application

As you integrate your application with the I/0 devices, controllers, and networks in your system:

- Refer to the user manual for your specific type of controller.
- Use this manual as a reference, when needed.

How to Use this Manual

This manual is divided into the basic tasks that you perform while programming a Logix5000 controller.

- Each chapter covers a task.
- The tasks are organized in the sequence that you will typically perform them.

As you use this manual, you will see some terms that are formatted differently from the rest of the text:

Text that is:	Identifies:	For example:	Means:
Italic	the actual name of an item that you see on your screen or in an example	Right-click *User-Defined* …	Right-click on the item that is named User-Defined.
bold	an entry in the "Glossary"	Type a **name** …	If you want additional information, refer to **name** in the "Glossary." If you are viewing the PDF file of the manual, click **name** to jump to the glossary entry.
`courier`	information that you must supply based on your application (a variable)	Right-click `name_of_program` …	You must identify the specific program in your application. Typically, it is a name or variable that you have defined.

Chapter 1

Manage Project Files

Create a Project File 1-1
Save Your Changes 1-2

Chapter 2

Organize Tasks

When to Use This Procedure 2-1
How to Use This Procedure. 2-1
Identify the Available Programming Languages. 2-1
Organize Your Logic 2-2
Verify the Controller 2-5

Chapter 3

Organize Tags

Plan Your Tags 3-1
Create a User-Defined Data Type. 3-8
Notes: 3-9
Create a Tag 3-10
Create Tags Using Microsoft® Excel® 3-11

Chapter 4

Program Routines

When to Use This Procedure 4-1
How to Use This Procedure. 4-1
Open the Routine 4-1
Enter Ladder Instructions. 4-3
Enter Function Block Instructions 4-4
Assign Operands. 4-7
Verify the Routine 4-10

Chapter 5

Access System Values

Monitor Status Flags 5-1
Get and Set System Data 5-2

Chapter 6

Assign Aliases

Alias Tags 6-1
Display Alias Information 6-2
Assign an Alias 6-3

Chapter 7

Assign an Indirect Address

When to Assign an Indirect Address. 7-1
Expressions 7-3

Chapter 8

Buffer I/O

When to Buffer I/O. 8-1
Buffer I/O. 8-1

Chapter 9

Test a Project

Test a Project 9-1
Configure a Communication Driver 9-1
Download a Project to the Controller 9-2
Select a Mode for the Controller 9-3
Correct Major Faults 9-4
Save Your Online Changes 9-4

Chapter 10

Communicate with Another Controller

When to Use This Procedure 10-1
How to Use This Procedure 10-1
Produce and Consume a Tag 10-1
What You Need To Do 10-3
Organize Tags for Produced or Consumed Data 10-3
Produce a Tag 10-4
Consume a Produced Tag 10-5
Produce Integers for a PLC-5C Controller 10-6
Produce REALs for a PLC-5C Controller 10-7
Consume Integers from a PLC-5C Controller 10-9
Adjust for Bandwidth Limitations 10-10
Send a Message 10-11

Chapter 11

Produce a Large Array

When to Use this Procedure 11-1
Produce a Large Array 11-2

Chapter 12

Communicate with an ASCII Device

When to Use this Procedure 12-1
How to Use This Procedure 12-1
Connect the ASCII Device 12-2
Configure the Serial Port 12-3
Configure the User Protocol 12-5
Create String Data Types 12-8
Read Characters from the Device 12-9
Send Characters to the Device 12-14
Enter ASCII Characters 12-21

Chapter 13

Process ASCII Characters
When to Use this Procedure 13-1
How to Use this Procedure 13-1
Extract a Part of a Bar Code. 13-2
Check the Bar Code Characters 13-4
Convert a Value 13-6
Decode an ASCII Message 13-8
Build a String 13-12

Chapter 14

Force Values
When to Force a Value 14-1
Enter a Force 14-2
Enter Forces from the Tags Window 14-2
Enter Forces from Ladder Logic 14-4
Enable Forces 14-5
Disable Forces 14-6
Remove Forces 14-6
Monitor Forces 14-7

Chapter 15

Develop a Fault Routine
When to Use This Procedure 15-1
Develop a Fault Routine 15-1
Test a Fault Routine 15-4

Chapter 16

Create a User-Defined Major Fault
When to Use this Procedure 16-1
Create a User-Defined Major Fault 16-1

Chapter 17

Monitor Minor Faults
When to Use This Procedure 17-1
Monitor Minor Faults 17-1

Chapter 18

Develop a Power-Up Routine
When to Use This Procedure 18-1
Develop a Power-Up Routine 18-1

Chapter 19

Store and Load a Project Using Nonvolatile Memory
When to Use This Procedure 19-1
How to Use This Procedure. 19-2
Store a Project. 19-3
Load a Project 19-6
Check for a Load 19-9
Clear Nonvolatile Memory 19-10

Appendix A

Fault Codes

When to Use This Appendix A-1
Major Fault Codes A-1
Minor Fault Codes A-3

Appendix B

IEC61131-3 Compliance

Using This Appendix B-1
Introduction B-1
Operating System B-2
Data Definitions B-2
Programming Languages B-3
Instruction Set B-4
IEC61131-3 Program Portability B-4
IEC Compliance Tables B-5

Glossary

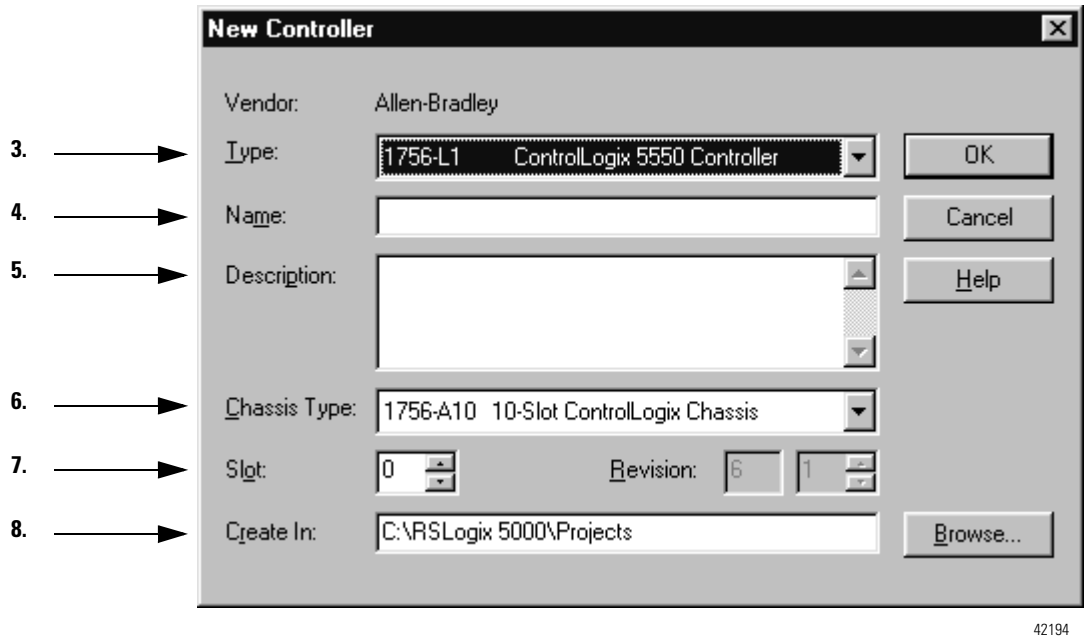
ASCII Character Codes

Manage Project Files

Create a Project File

Before you program a Logix5000 controller, you must create a **project file**:

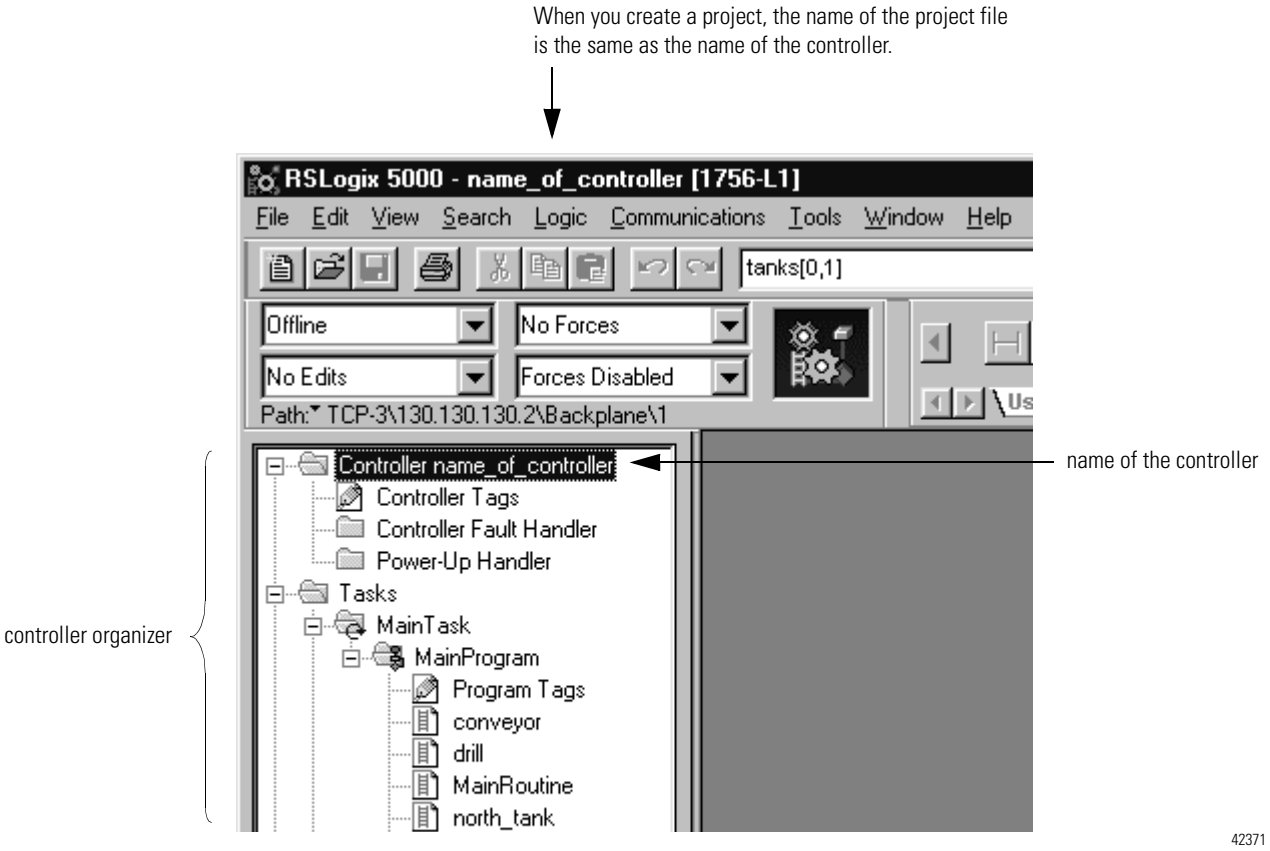
1. Start the RSLogix 5000™ software.
2. From the *File* menu, select *New*.



42194

3. Select the type of controller.
4. Type a **name** for the controller.
5. Type a description of the operations that the controller performs (optional).
6. Select the type of chassis (number of slots) that contains the controller (not applicable to some controllers).
7. Select or type the slot number where the controller is installed (not applicable to some controllers).
8. To store the file in a different folder (other than the default *Create In* path), click *Browse* and select a folder.

9. Click *OK*.



Save Your Changes

As you create logic and make configuration changes, save the project.

To:	Do this:
save your changes	From the *File* menu, select *Save*.
make a copy of the open project but keep the existing name of the controller	A. From the *File* menu, select *Save* As. B. Type a name for the project file. Use underscores [_] in place of spaces. C. Click *Save*.
make a copy of the project and assign a different name to the controller	A. From the *File* menu, select *Save* As. B. Type a name for the project file. Use underscores [_] in place of spaces. C. Click *Save*. D. In the controller organizer, right-click *Controller* `name_of_controller` folder and select *Properties*. E. Type a new name for the controller. F. Click *OK*.

Notes:

- Names **download** to the controller, while documentation (descriptions, rung comments) does not download to the controller.
- To change the name, chassis size, or slot number of the controller:
  a. In the controller organizer, right-click the *Controller* `name_of_controller` folder and select *Properties*.
  b. Change the required information.
  c. Click *OK*.

Notes:

Organize Tasks

When to Use This Procedure

After you create a project file, organize your project into tasks.

How to Use This Procedure

To organize your project into tasks, do the following steps:

- Identify the Available Programming Languages
- Organize Your Logic
- Verify the Controller

Identify the Available Programming Languages

Use the following table to identify which programming language you can use for your controller:

For this platform of controllers:	You can use this language:	
	ladder	function block
CompactLogix	✔	✔
ControlLogix	✔	✔
FlexLogix	✔	✔
SoftLogix	✔	

Notes:

- For controllers with multiple languages, you can mix languages in a project.
- To use function blocks, you must have the following catalog number of RSLogix 5000 software:
  - 9324-RLD700
- To see which components are installed with RSLogix 5000 software:
  1. Open RSLogix 5000 software.
  2. From the *Help* menu, choose *About RSLogix 5000*.

Organize Your Logic

To execute your logic, you use a **task** (s). There are two types of tasks:

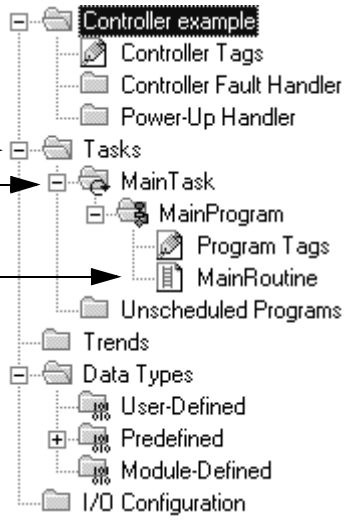
This type of task:	Will:
continuous task	continuously execute your logic (You can only have one continuous task.)
periodic task	• interrupt the continuous task • execute logic one time • return control to the continuous task (You can have more than one periodic task.)

The controller organizer shows the tasks of a controller.

The *Tasks* folder contains the tasks for the controller (i.e., your logic).

MainTask is the default continuous task. It runs all the time and repeatedly executes *MainProgram*.

Whenever *MainProgram* executes, any logic in *MainRoutine* will execute. You can use *MainRoutine* to call other routines (subroutines) within *MainProgram*.



42195

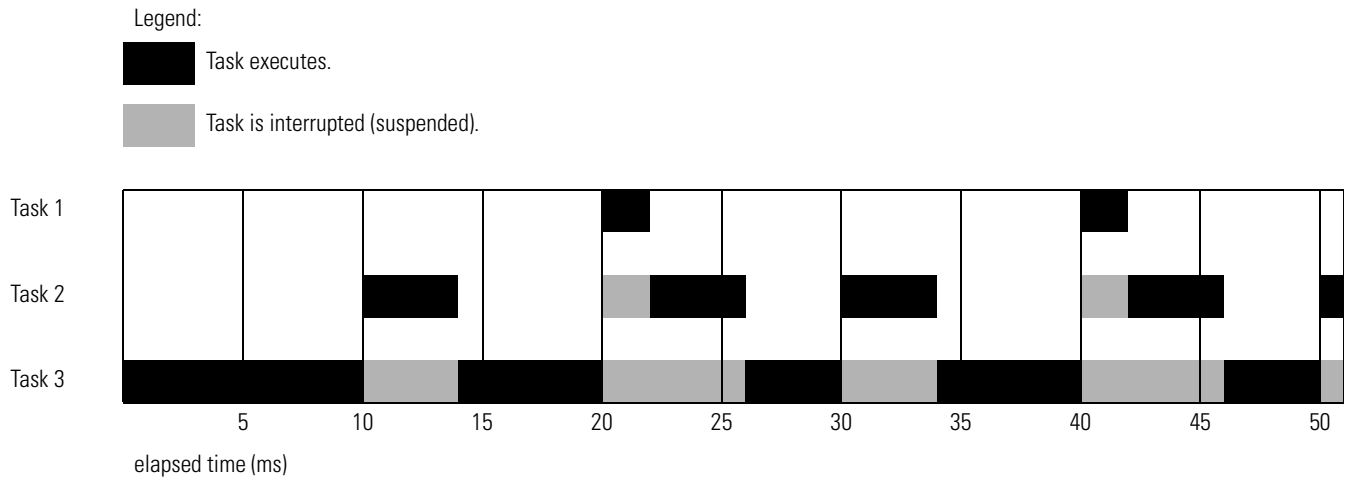
Select the task (s) for your logic:

If you execute:	Then:	Detailed steps:
a function at a constant rate (e.g., execute a PID loop every 100 ms) a function very fast	1. Create a periodic task for the function	A. In the controller organizer, right-click *Tasks* and select *New Task*. B. Type: • ***name***`_of_task` • ***description*** (optional) C. From the *Type* list, select *Periodic*. D. Under *Periodic Attributes*, type: • ***rate*** • ***priority*** E. Click *OK*.
	2. Create a program for the task.	A. Right-click `name_of_task` and select *New Program*. B. Type: • `name_of_program` • `description` (optional) C. Click *OK*.
	3. Create and assign a **main routine** (the routine to execute first in the program).	A. Click the + sign that is next to `name_of_task.` B. Right-click `name_of_program` and select *New Routine*. C. Type: • `name_of_main_routine` • `description` (optional) D. From the *Type* drop-down list, select the programming language for the routine. E. Click *OK*. F. Right-click `name_of_program` and select *Properties*. G. Click the *Configuration* tab. H. From the *Main* drop-down list, select `name_of_main_routine` I. Click *OK*. J. To add additional routines (subroutines) to the program, repeat steps B. to E.
	4. Place the remaining functions in *MainTask, MainProgram*. (See below.)	
multiple functions and use logic to decide when each function will execute	1. Create a routine (subroutine) for each function:	A. In the controller organizer, right-click *MainProgram* and select *New Routine*. B. Type the following properties for the routine (function): • ***name***`_of_routine` • ***description*** (optional) C. From the *Type* drop-down list, select the programming language for the routine. D. Click *OK*.
	2. Use a JSR instruction to call each subroutine.	
all functions all of the time	1. In *MainRoutine,* enter your logic. 2. Use rung comments to designate the different functions.	

The following example depicts the execution of a project with more than one task.

EXAMPLE Task execution order for a project with two periodic tasks and one continuous task

Task:	Task Type:	Priority Level:	Execution Time:
1	20ms periodic	5	2ms
2	10ms periodic	10	4ms
3	continuous	none (lowest)	24ms



Notes:

- All periodic tasks interrupt the continuous task.
- The highest priority task interrupts all lower priority tasks.
- A higher priority task can interrupt a lower priority task multiple times.
- When the continuous task completes a full scan, it restarts immediately.
- Tasks at the same priority execute on a time-slice basis at 1 ms intervals.
- To change the properties of a task, program, or routine (name, type, priority, etc.), right-click the task, program, or routine and select *Properties*.

Verify the Controller

As you program your project, periodically verify your work:

1. In the top-most toolbar of the RSLogix 5000 window, click
2. If any errors are listed at the bottom of the window:
   a. To go to the first error or warning, press the *F4* key.
   b. Correct the error according to the description in the Results window.
   c. Go to step 1.
3. To close the Results window, press the *Alt* + *1* keys.

Notes:

Organize Tags

Plan Your Tags

Logix5000 controllers store data in **tags** (in contrast to fixed data files that are numerically addressed). With tags, you can

- organize your data to mirror your machinery
- document (through tag names) your application as you develop it

When you create a tag, you assign the following properties:

Table 3.A Tag Properties

Property:	Description:
scope	defines which routines can access the data
name	identifies the data (Tags with different scopes can have the same name.)
data type	defines the organization of the data, such a bit, integer, or floating-point number

The following table outlines the most common data types and when to use each.

Table 3.B Data Types

For:	Select:
analog device in floating-point mode	REAL
analog device in integer mode (for very fast sample rates)	INT
ASCII characters	string
bit	BOOL
counter	COUNTER
digital I/O point	BOOL
floating-point number	REAL
integer (whole number)	DINT
sequencer	CONTROL
timer	TIMER

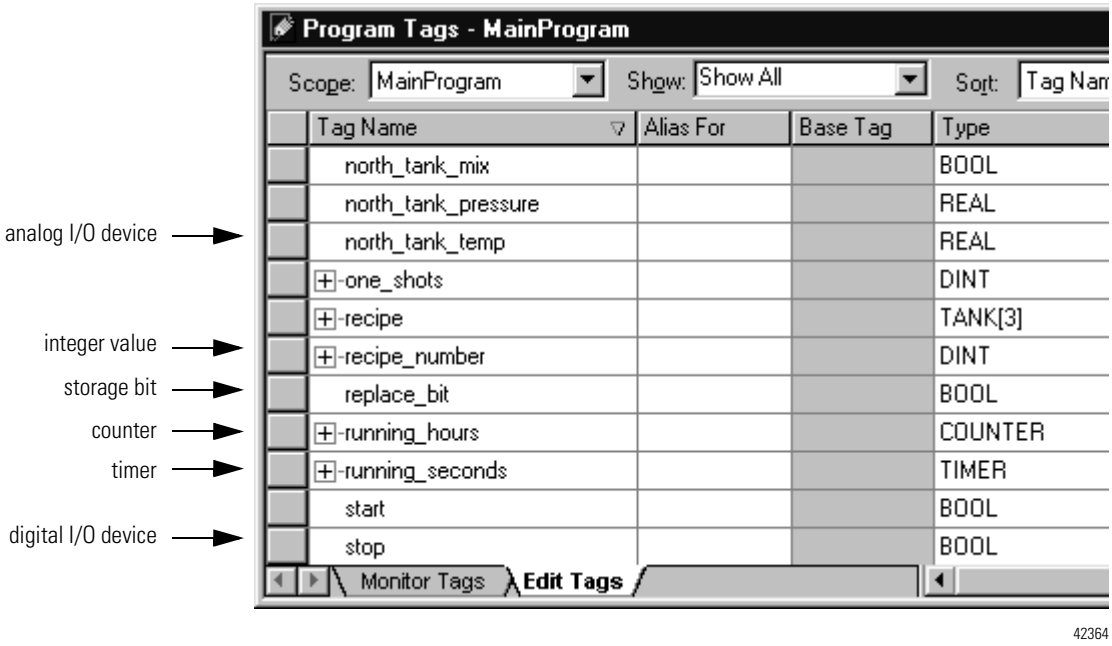
Use the following table to organize your data:

For a:	Use a:	Reference:
group of common attributes that are used by more than one machine	**user-defined data type**	Refer to "Create a User-Defined Data Type" on page 3-8.
group of data with the same data type	**array**	Refer to "Create a Tag" on page 3-10.
single value	tag of a single element	
I/O device		

The following examples show the different levels at which you can organize your data:

- Single element tags, on page 3-3
- Single dimension array, on page 3-4
- Two dimension array, on page 3-5
- User-defined data type that stores a recipe, on page 3-6
- User-defined data type that stores the data that is required to run a machine, on page 3-7

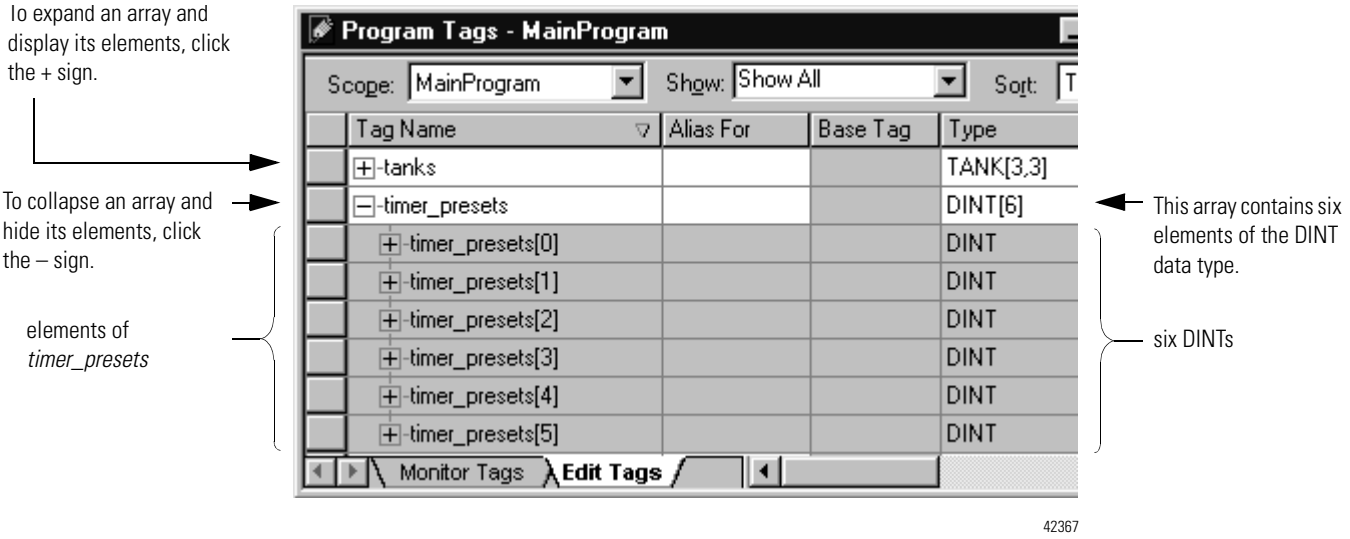
EXAMPLE Single element tags



EXAMPLE

Single dimension array

In this example, a single timer instruction times the duration of several steps. Each step requires a different preset value. Because all the values are the same data type (DINTs) an array is used.



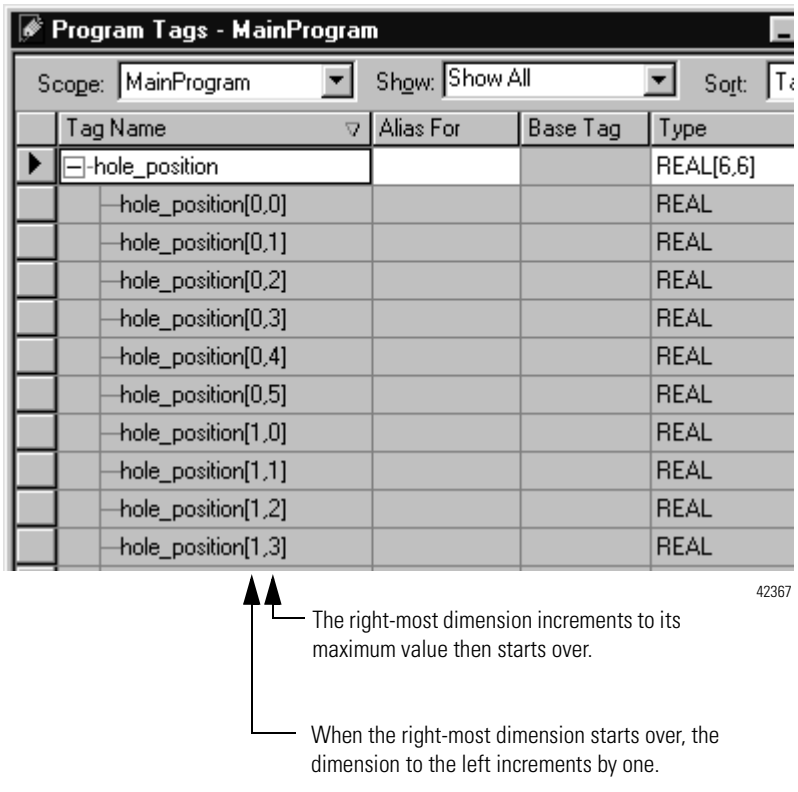
EXAMPLE

Two dimension array

A drill machine can drill one to five holes in a book. The machine requires a value for the position of each hole from the leading edge of the book. To organize the values into configurations, a two dimension array is used. The first subscript indicates the hole to which the value corresponds and the second subscript indications how many holes will be drilled (one to five).

	subscript of second dimension						Description
subscript of first dimension	0	1	2	3	4	5	
0							
1		1.5	2.5	1.25	1.25	1.25	Position of first hole from leading edge of book
2			8.0	5.5	3.5	3.5	Position of second hole from leading edge of book
3				9.75	7.5	5.5	Position of third hole from leading edge of book
4					9.75	7.5	Position of fourth hole from leading edge of book
5						9.75	Position of fifth hole from leading edge of book

In the Tags window, the elements are in the order depicted below.



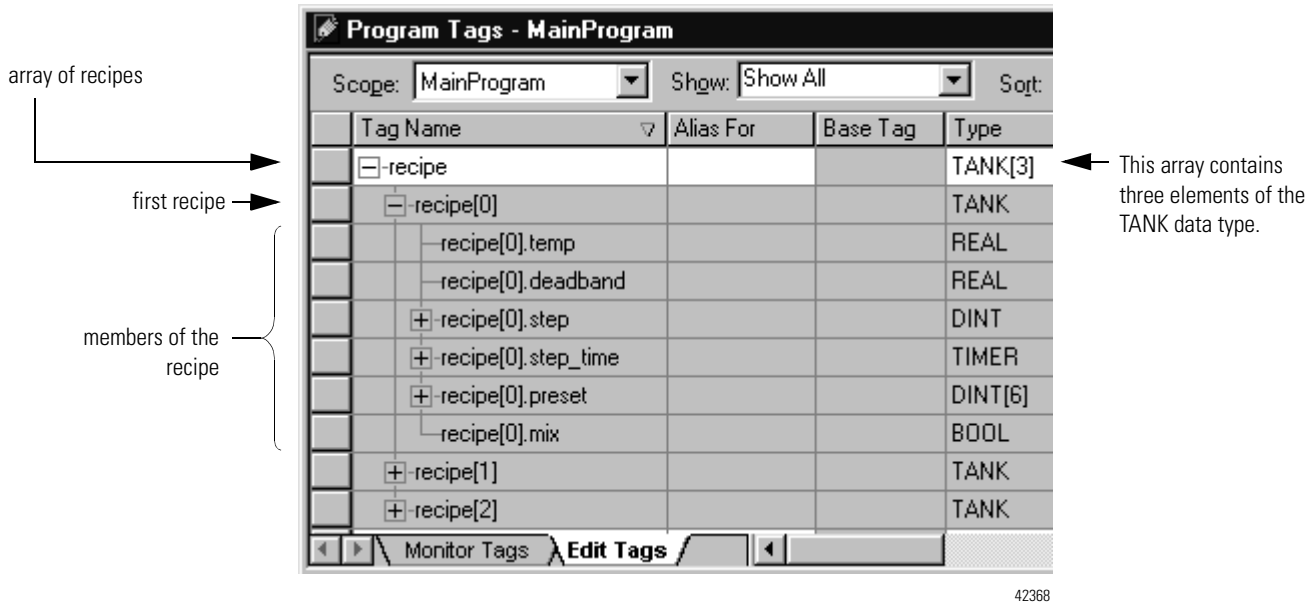
EXAMPLE

User-defined data type that stores a recipe

In a system of several tanks, each tank can run a variety of recipes. Because the recipe requires a mix of data types (REAL, DINT, BOOL, etc.) a user-defined data type is used.

Name (of data type): TANK	
Member Name	**Data Type**
temp	REAL
deadband	REAL
step	DINT
step_time	TIMER
preset	DINT[6]
mix	BOOL

An array that is based on this data type would look like this:



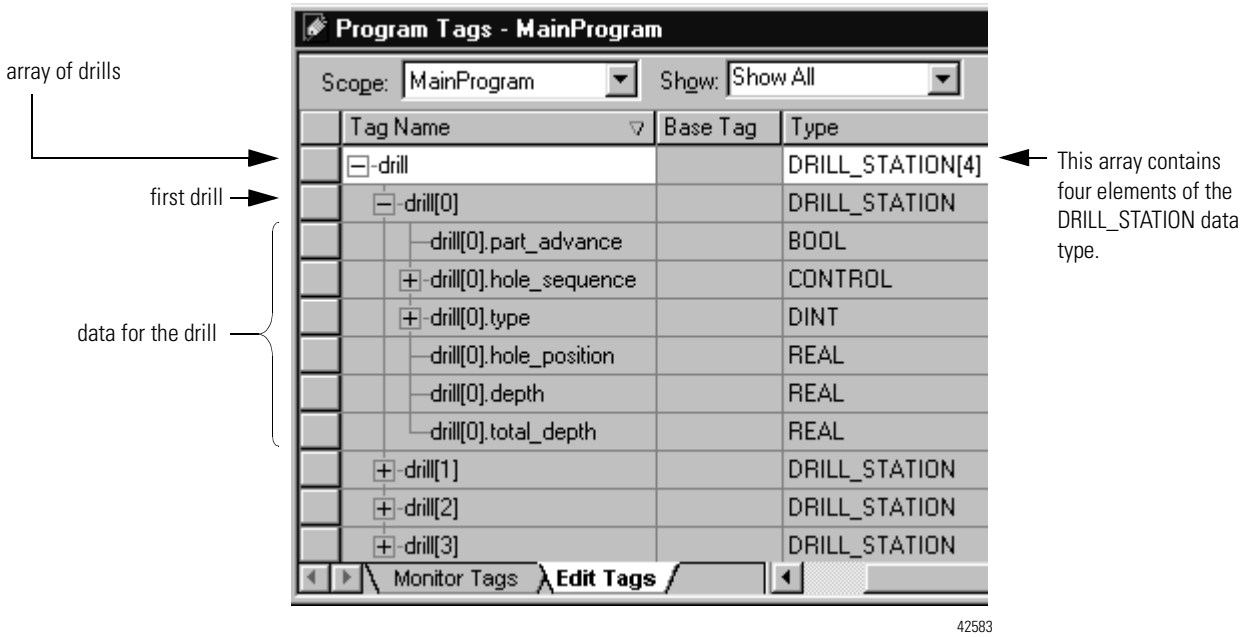
EXAMPLE

User-defined data type that stores the data that is required to run a machine

Because several drill stations require the following mix of data, a user-defined data type is created.

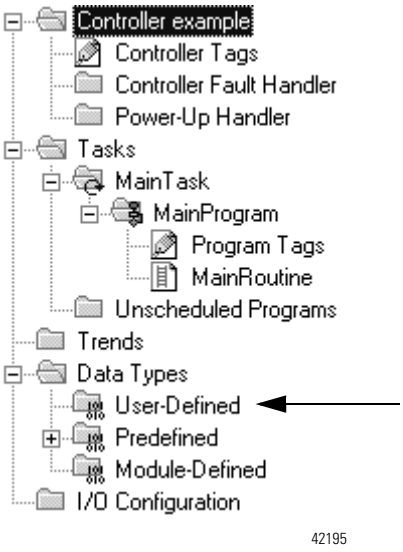
Name (of data type): DRILL_STATION	
Member Name	**Data Type**
part_advance	BOOL
hole_sequence	CONTROL
type	DINT
hole_position	REAL
depth	REAL
total_depth	REAL

An array that is based on this data type would look like this:

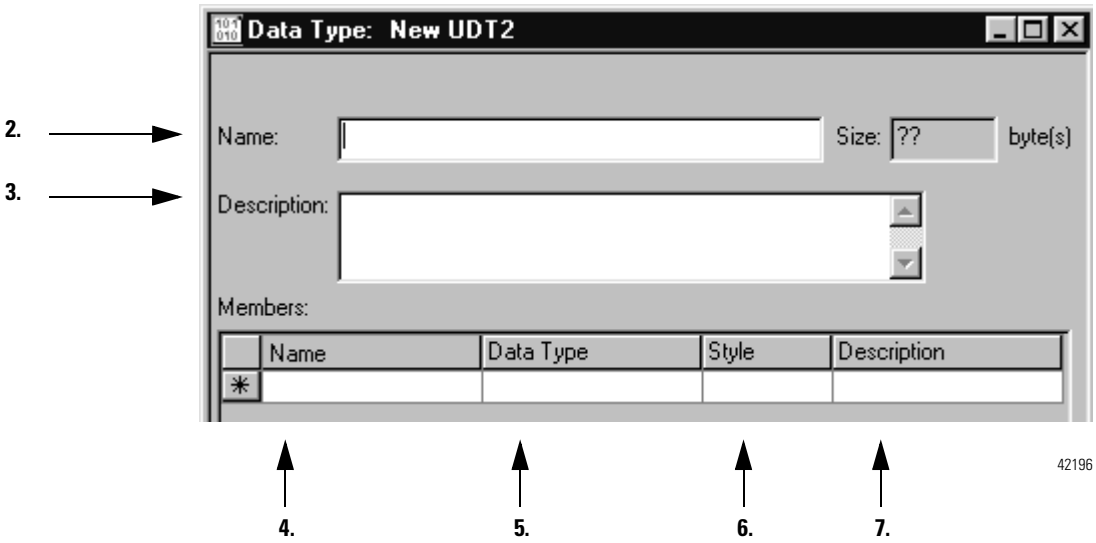


Create a User-Defined Data Type

To create a user-defined data type:



1. Right-click *User-Defined* and select *New Data Type*.



2. Type a **name** for the data type.
3. Type a **description** (optional).
4. Type the name of the first **member**.

5. Specify the data type for the member. See Table 3.B on page 3-1.

For an array, use the following format:

data_type[x]

where:

x is the number of elements in the array.

EXAMPLE If the member is an array of six DINTs, type DINT[6].

6. To display the value (s) of the member in a different **style** (radix), select the style.

7. Type a description for the member (optional).

8. Click *Apply*.

9. More members?

If:	Then:
Yes	Repeat steps 4. to 8.
No	Click *OK*.

Notes:

- If you include members that represent I/O devices, you must use ladder logic to copy the data between the members in the structure and the corresponding I/O tags. Refer to "Buffer I/O" on page 8-1.
- When you use the BOOL, SINT, or INT data types, place members that use the same data type in sequence:

more efficient

BOOL
BOOL
BOOL
DINT
DINT

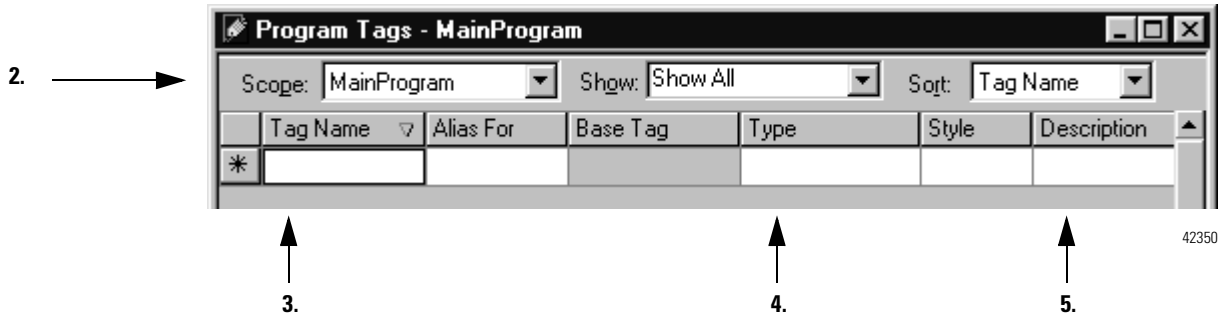
less efficient

BOOL
DINT
BOOL
DINT
BOOL

Create a Tag

To create a tag (including an array):

1. From the *Logic* menu, select *Edit Tags*.



2. Select a **scope** for the tag:

If you will use the tag:	Then select:
in more than one program within the project	*name_of_controller(controller)*
as a producer or consumer	
in a message	
in only one program within the project	program that will use the tag

3. Type a **name** for the tag.

4. Type the **data type**:

If the tag is:	Then type:
not an **array** (file)	*data_type*
one **dimension** array	*data_type[x]*
two dimension array	*data_type[x,y]*
three dimension array	*data_type[x,y,z]*

where:

data_type is the type of data that the tag or array stores. See Table 3.B on page 3-1.

x is the number of **elements** in the first dimension.

y is the number of elements in the second dimension.

z is the number of elements in the third dimension.

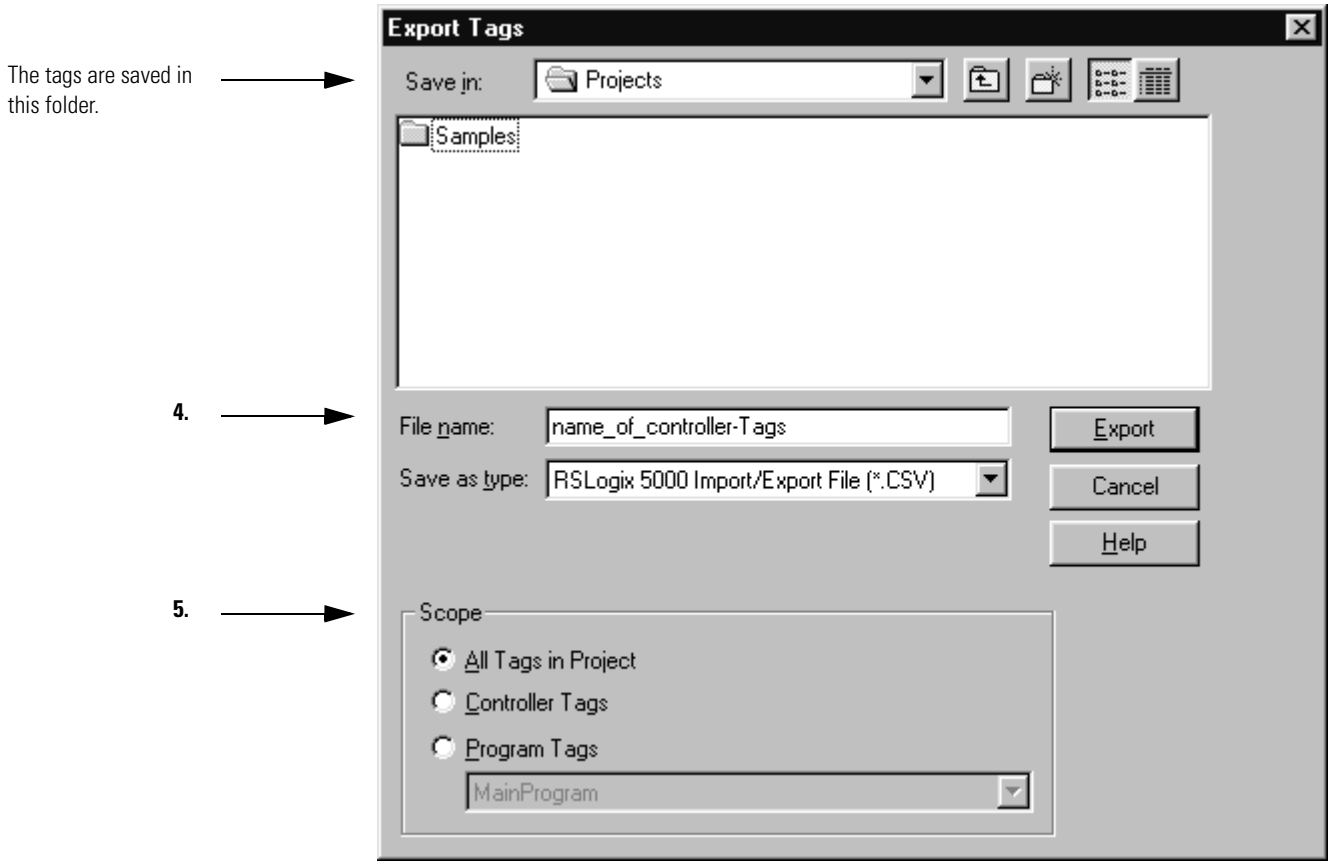
5. Type a **description** (optional).

Create Tags Using Microsoft® Excel®

You can also use spreadsheet software such as Microsoft Excel to create and edit tags. This lets you take advantage of the editing features in the spreadsheet software.

To create tags using Excel:

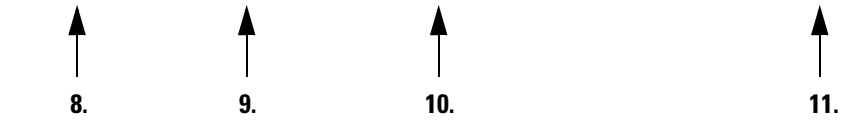
1. Open the RSLogix 5000 project.
2. Create several tags. (This helps to format the Excel spreadsheet.)
3. From the *Tools* menu, select *Export Tags*.



4. Note the name of the export file (*`project_name`-Tags*).
5. Select the scope of tags to export. If you select *Program Tags*, select the program tags to export.
6. Click *Export*.

7. In Microsoft Excel software, open the export file.

TYPE	SCOPE	NAME	DESCRIPTION	DATATYPE
TAG		in_cycle		DINT
TYPE	SCOPE	NAME	DESCRIPTION	DATATYPE
TAG	MainProgram	conveyor_alarm		BOOL
TAG	MainProgram	conveyor_on		BOOL
TAG	MainProgram	drill_1		DRILL_STATION
TAG	MainProgram	hole_position		REAL[6,6]
TAG	MainProgram	machine_on		BOOL



8. Enter TAG

9. Identify the scope of the tag:

If the scope is:	Then:
controller	Leave this cell empty.
program	Enter the name of the program

10. Enter the name of the tag.

11. Enter the data type of the tag.

12. Repeat steps 8. to 11. for each additional tag.

13. Save and close the file. (Keep it as a .CSV format.)

14. In the RSLogix 5000 software, from the *Tools* menu, select *Import Tags*.

15. Select the file that contains the tags and click *Import*.

The tags import into the project. The lower section of the RSLogix 5000 window displays the results.

Notes:

- You can configure tags to communicate directly with other controllers:

To:	Use a:
send data over the backplane and ControlNet network at a specified interval	**produced tag**
receive data from another controller over the backplane or ControlNet network at a specified interval	**consumed tag**

If you plan to use produced or consumed tags, you must follow additional guidelines as you organize your tags. Refer to "Communicate with Another Controller" on page 10-1.

- The following integer data types are also available:
  - **SINT** (8-bit integer)
  - **INT** (16-bit integer)

Typically, instructions convert SINT or INT values to an **optimal data type** (usually a DINT or REAL value) during execution. Because this requires additional time and memory, minimize the use of the SINT and INT data types.

Program Routines

When to Use This Procedure

After you organize your project into an initial set of routines and tags, use this procedure to develop the logic that each routine will execute.

How to Use This Procedure

To program a routine, do the following steps:

- Open the Routine
- Enter Ladder Instructions
- Enter Function Block Instructions
- Assign Operands
- Verify the Routine

Open the Routine

To close a folder and hide its contents (collapse), do one of the following:

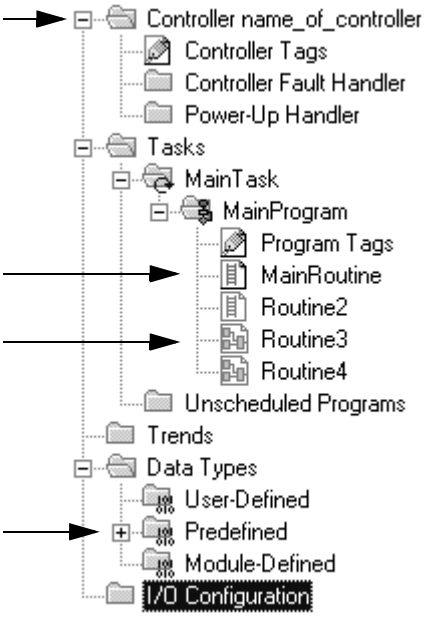
- Double-click the folder.
- Select the folder and press the ← key.
- Click the – sign.

To open a routine, double-click the routine.

If a routine is grayed-out, you cannot open the routine.

To open a folder and display its contents (expand), do one of the following:

- Double-click the folder.
- Select the folder and press the → key.
- Click the + sign.



Is the icon for the routine greyed out?

If the icon is:	**Then:**
not greyed-out	Double-click the routine.
greyed-out	You *cannot* open, program, or edit the routine. To determine the reason: 1. Double-click the routine. 2. At the bottom of the RSLogix 5000 window, what message does the status line display?

If:	**Then:**
"Failed to open the routine - editor not installed"	The function block editor is not installed. To install the function block editor, order the following catalog number of RSLogix 5000 software: • 9324-RLD700
"Source not available"	The source of the routine is unavailable. You can: • run the routine • display the properties of the routine • identify cross references to logic in the routine You *cannot*: • open (display) the routine • edit the routine • change the properties of the routine • search the routine • go to cross references within the routine • print the routine • export the routine

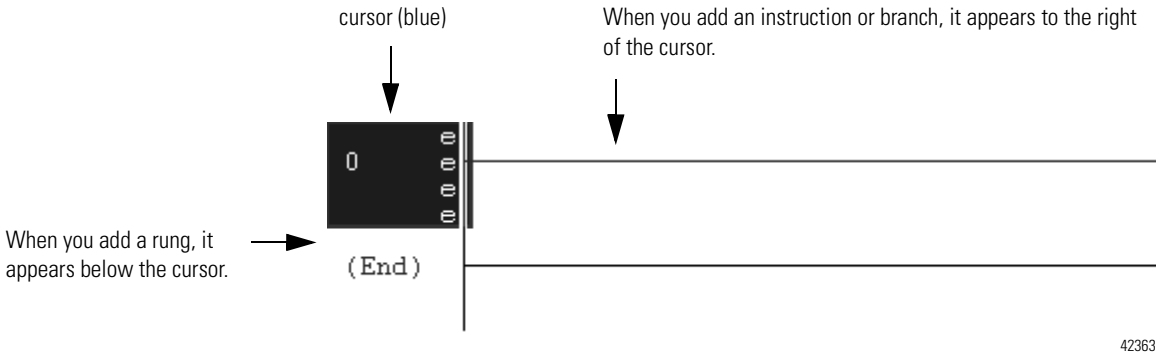
IMPORTANT If the source of a routine is unavailable, *do not* export the project.

- An export file (.L5K) contains only routines where the source code is available.
- If you export a project where the source code is *not* available for all routines, you will *not* be able to restore the entire project.

Enter Ladder Instructions

In an open ladder routine:

1. If the routine already contains logic, click where you want to enter your logic. (A new routine contains a rung that is ready for instructions.)


cursor (blue)
When you add an instruction or branch, it appears to the right of the cursor.
When you add a rung, it appears below the cursor.
0
(End)
42363

2. Add a ladder element:

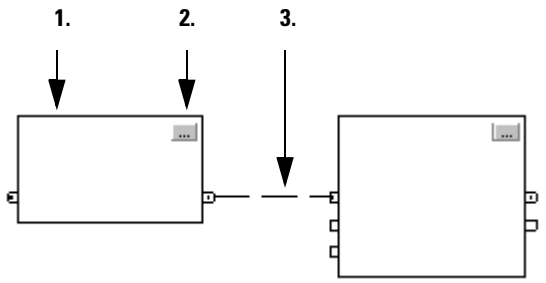
To add a:	Do this:
rung	Press the *Ctrl + R* keys.
instruction	A. Press the *Insert* key. B. Type the mnemonic for the instruction. C. Press the *Enter* key.
branch	A. Press the *Insert* key. B. Type BST. C. Press the *Enter* key. D. After you enter additional instructions, drag the right leg of the branch to the required location on the rung.

Enter Function Block Instructions

A function block routine contains the following elements:

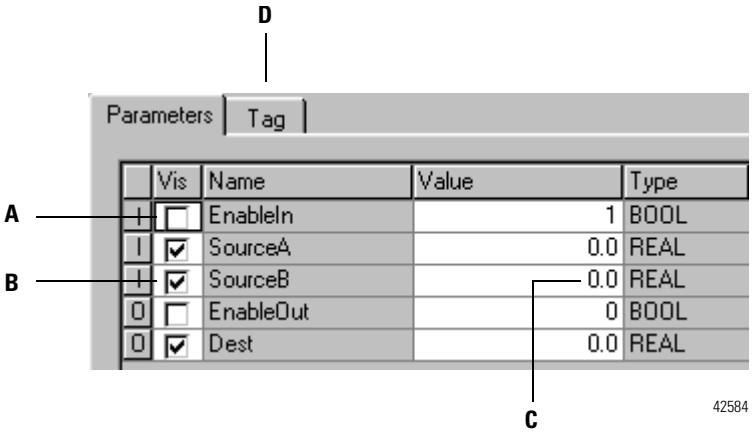
Element:	Purpose:
function block	Performs an operation on an input value (s) and produces an output value (s) • Pins on the left of the block are input pins. • Pins on the right of the block are output pins.
input reference (IREF)	Supplies a value from an input device or a tag in another routine or controller
output reference (OREF)	Supplies a value to an output device or a tag in another routine or controller
output wire connector (OCON)	Connects function blocks that are either far apart or on different sheets • Each OCON requires a unique name. • For each OCON, you must have at least one corresponding ICON (i.e., an ICON with the same name as the OCON). • Multiple ICONs can reference the same OCON. This lets you disperse data to several points in your routine.
input wire connector (ICON)	

To enter function block instructions, open a function block routine and complete the following steps:



42587

Step:	Detailed actions:
1. Enter the blocks that perform the required functions of the routine.	A. Press the *Insert* key. B. Type the mnemonic for the required block. C. Choose *OK*. D. Drag the block to a place on the diagram that makes the diagram easy to read. The location of a block does not affect the order in which the blocks execute.
2. Configure the properties of each block.	A. In the block, click ...

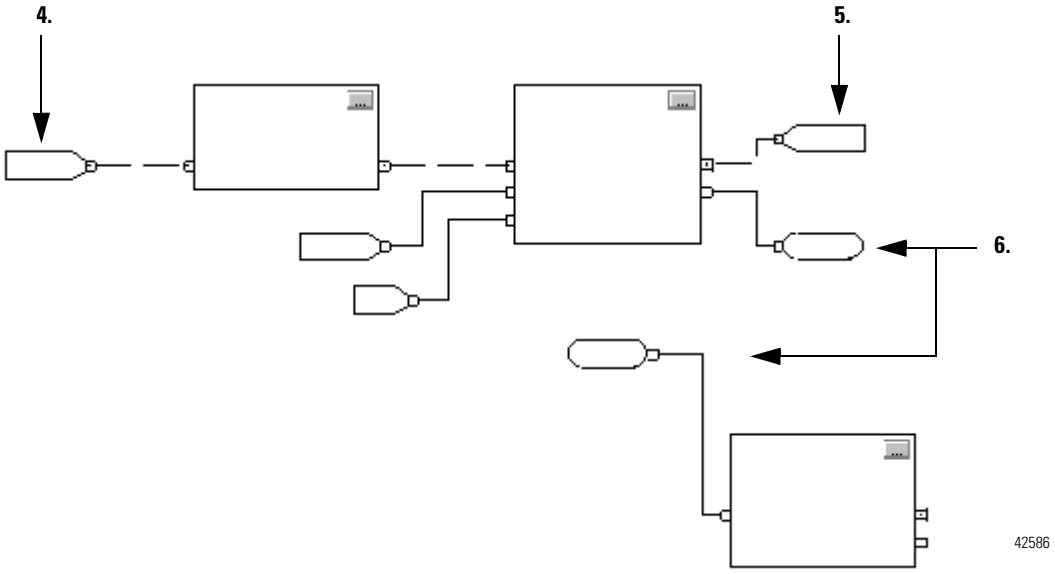


B. Edit the properties for the block:

To:	Do this:
show a pin for an operand	Select the check box next to the operand (A).
enter an immediate value	A. Clear (uncheck) the check box next to the operand (B). B. In the *Value* column, type the value for the operand (C).
change the name of the tag for the instruction	A. Click the Tag tab (D). B. Type a new name.

C. Choose *OK*.

3. Wire output pins to input pins.	Click an output pin and then click the required input pin. A green dot indicates a valid pin.



Step:	Detailed actions:
4. To supply a value from an input device or a tag, enter an input reference (IREF).	A. Press the *Insert* key. B. Choose *OK*. (IREF is the default selection.) C. Drag the IREF to the desired location, typically to the left of the block that uses the value. D. Click the pin of the IREF and then click the input pin that uses the value.
5. To supply a value to an output device or a tag, enter an output reference (OREF).	A. Press the *Insert* key. B. Type *OREF*. C. Choose *OK*. D. Drag the OREF to the desired location, typically to the right of the block that produces the output value. E. Click the output pin that supplies the value and then click the pin of the OREF.
6. To connect blocks that are far apart or on different sheets, enter an output wire connector (OCON) and an input wire connector (ICON).	A. Press the *Insert* key. B. Type *OCON*. C. Choose *OK*. D. Drag the OCON to the desired location, typically to the right of the block that produces the value. E. Click the output pin that produces the value and then click the pin of the OCON. F. Display the sheet that contains the block to which you want to connect. G. Press the *Insert* key. H. Type *ICON*. I. Choose *OK*. J. Drag the ICON to the desired location, typically to the left of the block that uses the value. K. Click the pin of the ICON and then click the input pin that uses the value.

Assign Operands

Each instruction requires one or more of the following:

- tag name
- value
- name of a routine, label, wire connector, etc.

The following table outlines the format for a tag name:

For a:	Specify:
tag	*tag_name*
bit number of an larger data type	*tag_name.bit_number*
member of a structure	*tag_name.member_name*
element of a one dimension array	*tag_name[x]*
element of a two dimension array	*tag_name[x,y]*
element of a three dimension array	*tag_name[x,y,z]*
element of an array within a structure	*tag_name.member_name[x]*
member of an element of an array	*tag_name[x,y,z].member_name*

where:

x is the location of the element in the first dimension.

y is the location of the element in the second dimension.

z is the location of the element in the third dimension.

For a structure within a structure, add an additional *.member_name* .

EXAMPLE Tags names

To access:	The tag name looks like this:
machine_on tag	machine_on
bit number 1 of the *one_shots* tag	one_shots.1
DN member (bit) of the *running_seconds* timer	running_seconds.DN
mix member of the *north_tank* tag	north_tank.mix
element 2 in the *recipe* array and element 1,1 in the *tanks* array	COP Copy File Source recipe[2] Dest tanks[1,1] Length 1
element 2 in the *preset* array within the *north_tank* tag	CLR Clear Dest north_tank.preset[2] 0
part_advance member of element 1 in the *drill* array	drill[1].part_advance

42357

To assign an immediate value, tag, wire connector, label, or similar item to an instruction:

1. Type or select the value or name of the item (e.g., tag name, wire connector name, label name):

<table>
<tr><th>If you want to:</th><th>For a:</th><th>Do this:</th></tr>
<tr><td rowspan="2">• specify an immediate value
• type the name of a tag, wire connector, label, or similar item</td><td>ladder instruction</td><td>A. Click the ? symbol.
B. Type the value or type the name of the tag, label, or similar item
C. Press the Enter key.</td></tr>
<tr><td>function block instruction</td><td>A. Click the ? symbol.
B. Click the ? symbol again.
C. Type the value or type the name of the tag or connector.
D. Press the Enter key.</td></tr>
<tr><td>select a tag, wire connector, label, or similar item from a list</td><td>→</td><td>A. Open the text entry box:
<table><tr><th>In a:</th><th>Do this:</th></tr><tr><td>ladder instruction</td><td>Double-click the ? symbol.</td></tr><tr><td>function block instruction</td><td>A. Click the ? symbol.
B. Click the ? symbol again.</td></tr></table>C. Click the ▼
D. Select a name:
<table><tr><th>To select a:</th><th>Do this:</th></tr><tr><td>wire connector, label, or similar type of name</td><td>Select the name.</td></tr><tr><td>tag</td><td>Double-click the tag name.</td></tr><tr><td>bit number</td><td>A. Click the tag name.
B. To the right of the tag name, click ▼
C. Click the required bit.</td></tr></table>D. Press the Enter key or click a different spot on the diagram.</td></tr>
<tr><td rowspan="2">drag a tag from the Tags window</td><td>ladder instruction</td><td>A. Locate the tag in the Tags window.
B. Click the tag two or three times until it highlights.
C. Drag the tag to its location on the instruction.</td></tr>
<tr><td>function block instruction</td><td>not available</td></tr>
</table>

2. Have you previously defined (created) this tag?

If:	Then:
Yes	Go to the next operand.
No	Create the tag: A. Right_click the tag and select *New "`tag_name.`"* (In older revisions of the software, the menu option is *Create "`tag_name.`"*) B. In the *Description* box, type a **description** for the tag (optional). C. In the *Data Type* box, type the **data type** for the tag:

If the tag is:	Then type:
not an **array** (file)	*`data_type`*
one **dimension** array	*`data_type[x]`*
two dimension array	*`data_type[x,y]`*
three dimension array	*`data_type[x,y,z]`*

where:
`data_type` is the type of data that the tag or array stores. See Table 3.B on page 3-1.
`x` is the number of **elements** in the first dimension.
`y` is the number of elements in the second dimension.
`z` is the number of elements in the third dimension.

D. From the *Scope* list, select the **scope** of the tag:

If you will use the tag:	Then select:
in more than one program within the project	*`name_of_controller`*(controller)
as a producer or consumer	
in a message	
in only one program within the project	program that will use the tag

E. Choose *OK*.

Verify the Routine

As you program your routine (s), periodically verify your work:

1. In the top-most toolbar of the RSLogix 5000 window, click

2. If any errors are listed at the bottom of the window:
a. To go to the first error or warning, press the *F4* key.
b. Correct the error according to the description in the Results window.
c. Go to step 1.

3. To close the Results window, press the *Alt + 1* keys.

Access System Values

Monitor Status Flags

The controller supports status keywords you can use in your logic to monitor specific events:

- The status keywords are *not* case sensitive.
- Because the status flags can change so quickly, RSLogix 5000 software does *not* display the status of the flags. (I.e., Even when a status flag is set, an instruction that references that flag is *not* highlighted.)
- You *cannot* define a tag alias to a keyword.

You can use these key words:

To determine if:	Use:
the value you are storing cannot fit into the destination because it is either: • greater than the maximum value for the destination • less than the minimum value for the destination **Important:** Each time S:V goes from cleared to set, it generates a minor fault (type 4, code 4)	S: V
the instruction's destination value is 0	S:Z
the instruction's destination value is negative	S:N
an arithmetic operation causes a carry or borrow that tries to use bits that are outside of the data type For example: • adding 3 + 9 causes a carry of 1 • subtracting 25 - 18 causes a borrow of 10	S:C
this is the first, normal scan of the routines in the current program	S:FS
at least one minor fault has been generated: • The controller sets this bit when a minor fault occurs due to program execution. • The controller does not set this bit for minor faults that are not related to program execution, such as battery low.	S:MINOR

Get and Set System Data

The controller stores system data in objects. There is no status file, as in the PLC-5 controller. Use the GSV/SSV instructions get and set controller system data that is stored in objects:

- The GSV instruction retrieves the specified information and places it in the destination.
- The SSV instruction sets the specified attribute with data from the source.

ATTENTION

Use the SSV instruction carefully. Making changes to objects can cause unexpected controller operation or injury to personnel.

To get or set a system value:

1. Open the RSLogix 5000 project.
2. From the *Help* menu, select *Contents*.
3. Click the *Index* tab.
4. Type *gsv/ssv objects* and click *Display*.

5. Click the required object.

To get or set:	Click:
axis of a servo module	AXIS
system overhead timeslice	CONTROLLER
physical hardware of a controller	CONTROLLERDEVICE
coordinated system time for the devices in one chassis	CST
DF1 communication driver for the serial port	DF1
fault history for a controller	FAULTLOG
attributes of a message instruction	MESSAGE
status, faults, and mode of a module	MODULE
group of axes	MOTIONGROUP
fault information or scan time for a program	PROGRAM
instance number of a routine	ROUTINE
configuration of the serial port	SERIALPORT
properties or elapsed time of a task	TASK
wall clock time of a controller	WALLCLOCKTIME

6. In the list of attributes for the object, identify the attribute that you want to access.

7. Create a tag for the value of the attribute:

If the data type of the attribute is:	Then:
one element (e.g., DINT)	Create a tag for the attribute.
more than one element (e.g., DINT[7])	A. Create a user-defined data type that matches the organization of data that is used by the attribute. B. Create a tag for the attribute and use the data type from Step A..

8. In your ladder logic routine, enter the appropriate instruction:

To:	Enter this instruction:
get the value of an attribute	GSV
set the value of an attribute	SSV

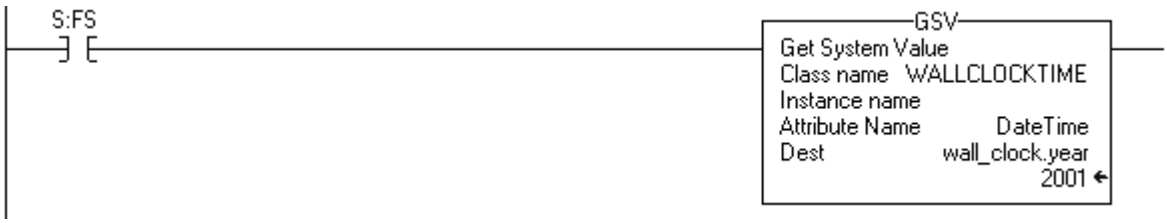
9. Assign the required operands to the instruction:

For this operand:	Select:
Class name	name of the object
Instance name	name of the specific object (e.g., name of the required I/O module, task, message) • Not all objects require this entry. • To specify the current task, program, or routine, select *THIS*.
Attribute Name	name of the attribute
Dest (GSV)	tag that will store the retrieved value • If the tag is a user-defined data type or an array, select the first member or element.
Source (SSV)	tag that stores the value to be set • If the tag is a user-defined data type or an array, select the first member or element.

The following examples gets the current date and time.

EXAMPLE Get a system value

At the first scan, gets the *DateTime* attribute of the *WALLCLOCKTIME* object and stores it in the *wall_clock* tag, which is based on a user-defined data type.



42370

For more information, see the *Logix5000 Controllers General Instruction Set Reference Manual*, publication 1756-RM003.

Assign Aliases

Alias Tags

An alias tag lets you create one tag that represents another tag.

- Both tags share the same value (s).
- When the value (s) of one of the tags changes, the other tag reflects the change as well.

Use aliases in the following situations:

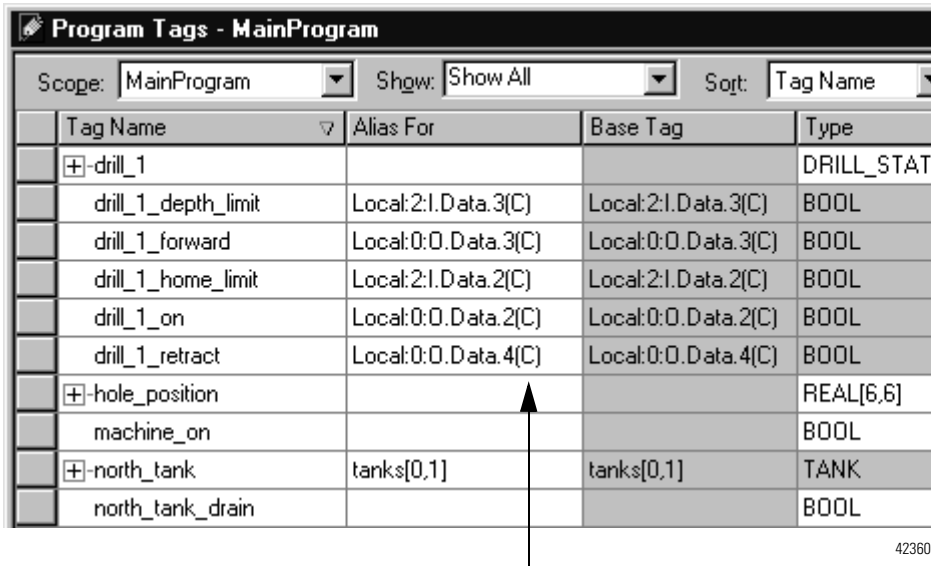
- program logic in advance of wiring diagrams
- assign a descriptive name to an I/O device
- provide a more simple name for a complex tag
- use a descriptive name for an element of an array

The tags window displays alias information.

drill_1_depth_limit is an alias for *Local:2:I.Data.3* (a digital input point). When the input turns on, the alias tag also turns on.

drill_1_on is an alias for *Local:0:O.Data.2* (a digital output point). When the alias tag turns on, the output tag also turns on.

north_tank is an alias for *tanks[0,1]*.

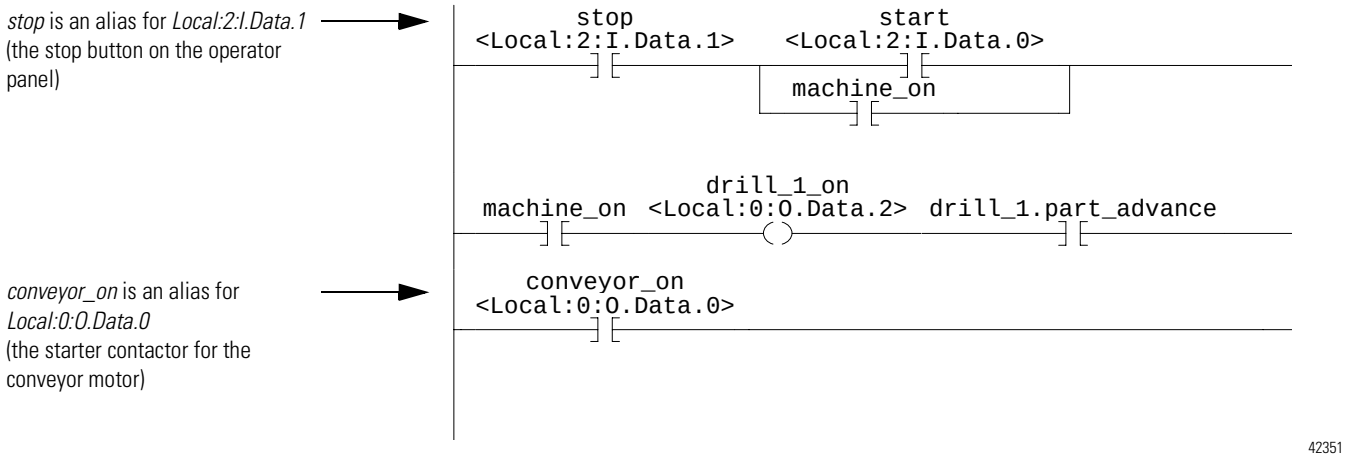


The *(C)* indicates that the tag is at the controller scope.

A common use of alias tags is to program logic before wiring diagrams are available:

1. For each I/O device, create a tag with a name that describes the device, such as *conveyor* for the conveyor motor.
2. Program your logic using the descriptive tag names. (You can even test your logic without connecting to the I/O.)
3. Later, when wiring diagrams are available, add the I/O modules to the I/O configuration of the controller.
4. Finally, convert the descriptive tags to aliases for their respective I/O points or channels.

The following logic was initially programmed using descriptive tag names, such as *stop* and *conveyor_on*. Later, the tags were converted to aliases for the corresponding I/O devices.



Display Alias Information

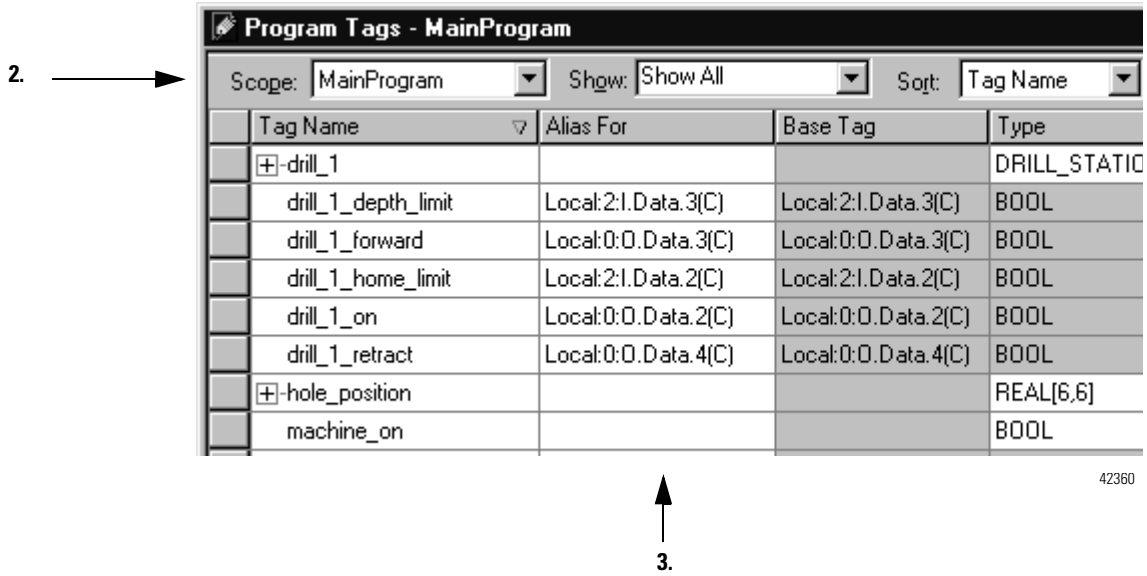
To show (in your logic) the tag to which an alias points:

1. From the *Tools* menu, select *Options*.
2. Click the *Ladder Display* tab.
3. Select the *Show Tag Alias Information* check box.
4. Click *OK*.

Assign an Alias

To assign a tag as an **alias tag** for another tag:

1. From the *Logic* menu, select *Edit Tags*.



2. Select the **scope** of the tag.
3. To the right of the tag name, click the *Alias For* cell.

   The cell displays a ▼
4. Click the ▼
5. Select the tag that the alias will represent:

To:	Do this:
select a tag	Double-click the tag name.
select a bit number	A. Click the tag name. B. To the right of the tag name, click ▼ C. Click the required bit.

6. Press the *Enter* key or click another cell.

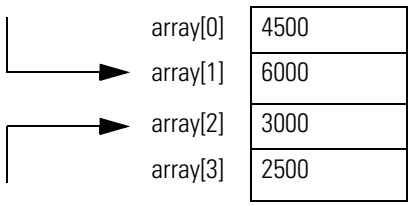
Notes:

Assign an Indirect Address

When to Assign an Indirect Address

If you want an instruction to access different elements in an array, use a tag in the subscript of the array (an indirect address). By changing the value of the tag, you change the element of the array that your logic references.

When *index* equals 1, *array[index]* points here.



When *index* equals 2, *array[index]* points here.

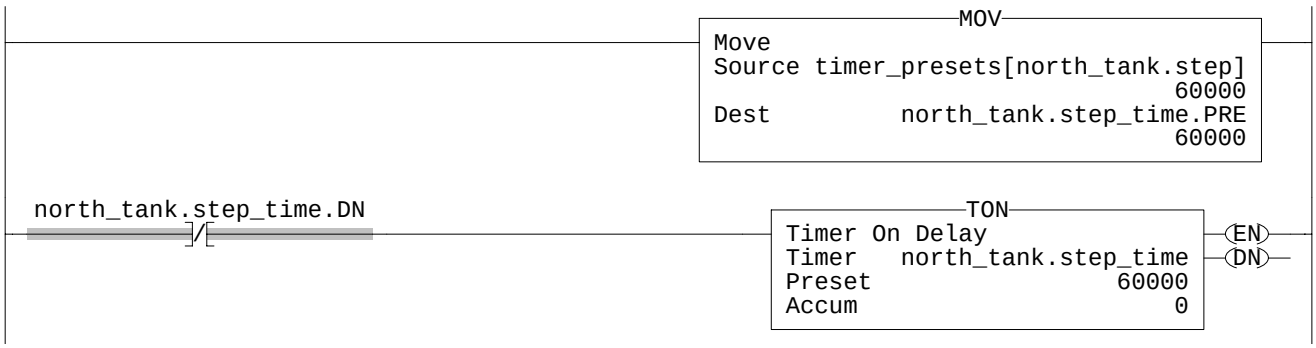
The following table outlines some common uses for an indirect address:

To:	Use a tag in the subscript and:
select a recipe from an array of recipes	Enter the number of the recipe in the tag.
load a specific machine setup from an array of possible setups	Enter the desired setup in the tag.
load parameters or states from an array, one element at a time	A. Perform the required action on the first element. B. Use an ADD instruction to increment the tag value and point to the next element in the array.
log error codes	
perform several actions on an array element and then index to the next element	

The following example loads a series of preset values into a timer, one value (array element) at a time.

EXAMPLE Step through an array

The *timer_presets* array stores a series of preset values for the timer in the next rung. The *north_tank.step* tag points to which element of the array to use. For example, when *north_tank.step* equals 0, the instruction loads *timer_presets[0]* into the timer (60,000 ms).



When *north_tank.step_time* is done, the rung increments *north_tank.step* to the next number and that element of the *timer_presets* array loads into the timer.

When *north_tank.step* exceeds the size of the array, the rung resets the tag to start at the first element in the array. (The array contains elements 0 to 3.)

42358

Expressions

You can also use an expression to specify the subscript of an array.

- An expression uses operators, such as + or -, to calculate a value.
- The controller computes the result of the expression and uses it as the array subscript.

You can use these operators to specify the subscript of an array:

Operator:	Description:
+	add
-	subtract/negate
*	multiply
/	divide
ABS	Absolute value
AND	AND
FRD	BCD to integer

Operator:	Description:
MOD	Modulo
NOT	complement
OR	OR
SQR	square root
TOD	integer to BCD
TRN	Truncate
XOR	exclusive OR

Format your expressions as follows:

If the operator requires:	Use this format:	Examples:
one value (tag or expression)	`operator(value)`	ABS(*tag_a*)
two values (tags, constants, or expressions)	`value_a operator value_b`	• *tag_b* + 5 • *tag_c* AND *tag_d* • (*tag_e* ** 2) MOD (*tag_f* / *tag_g*)

Notes:

Buffer I/O

When to Buffer I/O

To buffer I/O is a technique in which logic does not directly reference or manipulate the tags of real I/O devices. Instead, the logic uses a copy of the I/O data. Buffer I/O in the following situations:

- To prevent an input or output value from changing during the execution of a program. (I/O updates **asynchronous** to the execution of logic.)
- To copy an input or output tag to a member of a structure or element of an array.

Buffer I/O

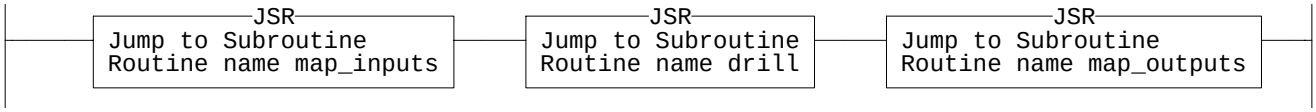
To buffer I/O, perform these actions:

1. On the rung before the logic for the function (s), copy or move the data from the required input tags to their corresponding buffer tags.
2. In the logic of the function (s), reference the buffer tags.
3. On the rung after the function (s), copy the data from the buffer tags to the corresponding output tags.

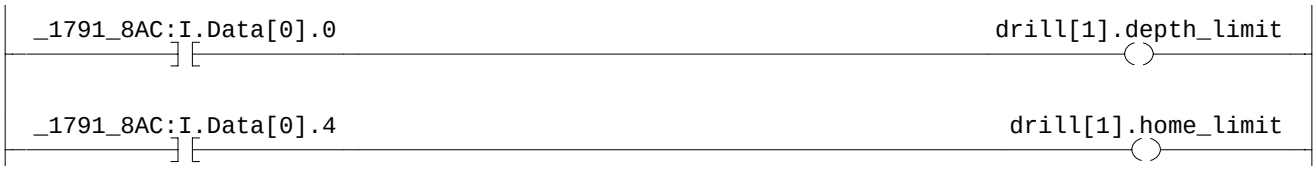
The following example copies inputs and outputs to the tags of a structure for a drill machine.

EXAMPLE Buffer I/O

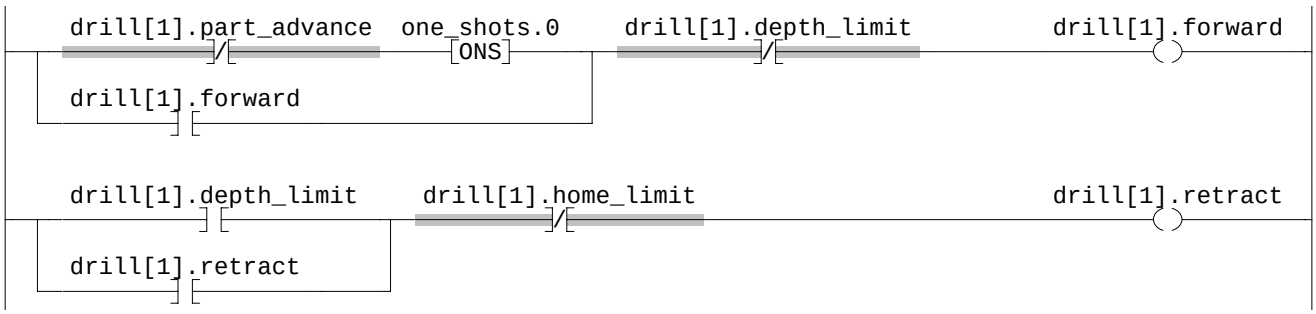
The main routine of the program executes the following subroutines in this sequence.



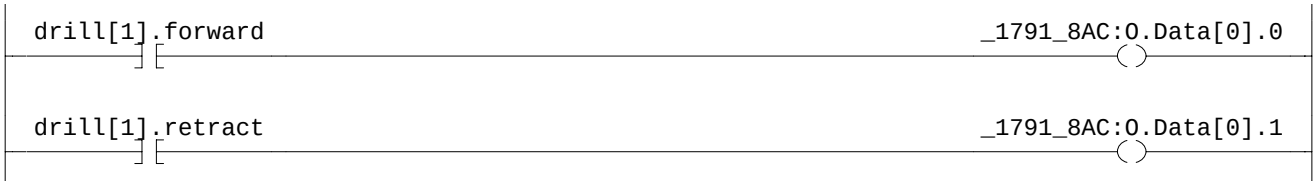
The *map_inputs* routine copies the values of input devices to their corresponding tags that are used in the *drill* routine.



The *drill* routine executes the logic for the drill machine.



The *map_outputs* routine copies the values of output tags in the *drill* routine to their corresponding output devices.



42369

The following example uses the CPS instruction to copy an array of data that represent the input devices of a DeviceNet network.

EXAMPLE Buffer I/O

Local:0:I.Data stores the input data for the DeviceNet network that is connected to the 1756-DNB module in slot 0. To synchronize the inputs with the application, the CPS instruction copies the input data to *input_buffer*.

- While the CPS instruction copies the data, no I/O updates can change the data.
- As the application executes, it uses for its inputs the input data in *input_buffer*.

42578

Notes:

Test a Project

Test a Project

To test a project, complete these actions:

- Configure a Communication Driver
- Download a Project to the Controller
- Select a Mode for the Controller
- Correct Major Faults
- Save Your Online Changes

In addition, you may perform these actions:

- Use program control instructions to isolate logic execution to specific routines or rungs. See *Logix5000 Controllers General Instruction Set Reference Manual*, publication 1756-RM003.
- Force input or output values. Refer to "Force Values" on page 14-1.

Configure a Communication Driver

The RSLogix 5000 software requires a communication driver to communicate with a controller. You configure communication drivers using RSLinx software:

1. Start RSLinx™ software.
2. From the *Communications* menu, select *Configure Drivers*.
3. From the Available Driver Types drop-down list, select a driver:

For this type of communication card:	Select this type of driver:
serial	RS-232 DF1 Devices
DH+™	1784-KT/KTX(D)/PKTX(D)
ControlNet™	1784-KTC(X)
Ethernet™	Ethernet devices

4. Click *Add New*.
5. If you want assign a descriptive name to the driver, change the default name.
6. Choose *OK*.

7. Configure the driver:

For this driver:	Do this:
serial	A. From the *Comm Port* drop-down list, select the serial port that the driver will use. B. From the Device drop-down list, select *Logix 5550-Serial Port*. C. Click *Auto-Configure*.
ControlNet	A. In the *Station Name* box, type a name that will identify the computer in the RSWho window. B. Select the interrupt value, memory address, and I/O base address. C. In the *Net Address* box, type the ControlNet node number that you want to assign to the computer.
DH+	A. From the *Value* drop-down list, select the type of interface card that the driver will use. B. In the *Property* list, select the next item. C. In the *Value* box, type or select the appropriate value. D. Repeat steps B. and C. for the remaining properties.
Ethernet	For each Ethernet device on this network with which you want to communicate (e.g., each 1756-ENET module or PLC-5E controller), add a map entry: A. In the *Host Name* column, type the IP address or host name of the Ethernet device. B. Do you want to communicate with another Ethernet device on this network? **If:** / **Then:** Yes — 1. Choose *Add New*. 2. Go to Step A. No — Go to the next step.

3. Click *OK*.

4. Click *Close*.

Download a Project to the Controller

1. Open the RSLogix 5000 project that you want to download.

2. From the Communications menu, select *Who Active*.

3. Click the + sign next to the network that you want to use for the download.

4. Continue expanding the network until the dialog box shows the controller to which you want to download.

5. Select the controller and click *Download*.

6. To confirm the download, click *Download*.

Select a Mode for the Controller

To test a project, select a mode for the controller:

Select the required mode.

If you want to:	Select one of these modes:				
	Run	Program	Remote		
			Run	Program	Test
turn outputs to the state commanded by the logic of the project	✔		✔		
turn outputs to their configured state for Program mode		✔		✔	✔
execute (scan) tasks	✔		✔		✔
change the mode of the controller through software			✔	✔	✔
download a project		✔	✔	✔	✔
schedule a ControlNet network		✔		✔	
while online, edit the project		✔	✔	✔	✔
send messages	✔		✔		✔
send and receive data in response to a message from another controller	✔	✔	✔	✔	✔
produce and consume tags	✔	✔	✔	✔	✔

Correct Major Faults

If the controller enters the **faulted mode**, a **major fault** occurred and the controller stopped executing the logic.

If the controller status box is red and displays *Faulted*, a major fault occurred and the controller is no longer executing the logic.

To correct a major fault:

1. From the *Communications* menu, select *Go To Faults*.
2. Use the information in the *Recent faults list* to correct the cause of the fault. Refer to "Major Fault Codes" on page A-1.
3. Click the *Clear Majors* button.

TIP

You can also clear a major fault by using the keyswitch on the controller. Turn the keyswitch to *Prog*, then to *Run*, and then back to *Prog*.

Save Your Online Changes

If you make changes to the project while **online**, save the project so that the offline project file matches the online project file:

If you want to:	Do this:
save online changes *and* data values	From the *File* menu, select *Save*.
save online changes but *not* online data values	A. From the *Communications* menu, select *Go Offline*. B. From the *File* menu, select *Save*.

Communicate with Another Controller

When to Use This Procedure

Use this procedure to transfer data between controllers (send or receive data). You can transfer data using either of these methods:

- Produce and Consume a Tag
- Send a Message

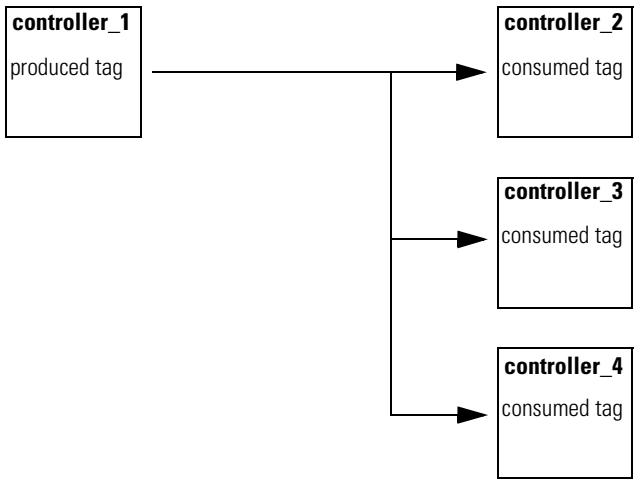
How to Use This Procedure

Select a method for transferring data between controllers:

If the data:	Then:	See page:
needs regular delivery at a rate that you specify (i.e., deterministic)	Produce and Consume a Tag	10-1
is sent when a specific condition occurs in your application	Send a Message	10-11

Produce and Consume a Tag

A **produced tag** sends its data to one or more **consumed tags** (consumers) without using ladder logic.



You can use produced and consumed tags with the following controller and network combinations.

This controller:	Can produce and consume tags over this network:		
	Backplane	ControlNet	Ethernet
PLC-5		✔	
ControlLogix	✔	✔	✔
FlexLogix		✔	
SoftLogix		✔	

Produced and consumed tags work as follows:

- A **connection** transfers the data between controllers:
  - Multiple controllers can consume (receive) the data.
  - The data updates at the **requested packet interval (RPI)**, as configured by the consuming tags.
- Each produced or consumed tag uses the following number of connections:

Each:	Uses this many connections:
produced tag	*number_of_consumers* + 1
consumed tag	1

EXAMPLE

Connections used by produced or consumed tags

- Producing a tag for 5 controllers (consumers) uses 6 connections (5 consumers + 1 = 6).
- Producing 4 tags for 1 controller uses 8 connections:
  - Each tag uses 2 connections (1 consumer + 1 = 2).
  - 2 connections per tag x 4 tags = 8 connections
- Consuming 4 tags from a controller uses 4 connections (1 connection per tag x 4 tags = 4 connections).

What You Need To Do

To share data with another controller (s), perform these actions:

- Organize Tags for Produced or Consumed Data
- Produce a Tag
- Consume a Produced Tag

Depending on your system, you may also have to perform these actions:

- Produce Integers for a PLC-5C Controller
- Produce REALs for a PLC-5C Controller
- Consume Integers from a PLC-5C Controller
- Adjust for Bandwidth Limitations

Organize Tags for Produced or Consumed Data

As you create tags that will eventually produce or consume data (share data), follow these guidelines:

1. Create the tags at the **controller scope**. You can only share controller-scoped tags.

2. Use one of these data types:
   - DINT
   - REAL
   - array of DINTs or REALs
   - user-defined

3. To share a data type other than those listed in guideline 2., create a user-defined data type that contains the required data.

4. Use the same data type for the produced tag and corresponding consumed tag (s).

5. To share tags with a PLC-5C controller, use a user-defined data type. See these sections:
   - Produce Integers for a PLC-5C Controller, 10-6
   - Produce REALs for a PLC-5C Controller, 10-7
   - Consume Integers from a PLC-5C Controller, 10-9

6. Limit the size of the tag to less than or equal to 500 bytes. If you must transfer more than 500 bytes, create logic to transfer the data in packets. Refer to "Produce a Large Array" on page 11-1.

7. If you share the tag over a ControlNet network, the tag may need to be less than 500 bytes. Refer to "Adjust for Bandwidth Limitations" on page 10-10.

8. If you are producing several tags for the same controller:
   - Group the data into one or more user-defined data types. (This uses less connections than producing each tag separately.)
   - Group the data according to similar update rates. (To conserve network bandwidth, use a greater RPI for less critical data.)

   For example, you could create one tag for data that is critical and another tag for data that is not as critical.

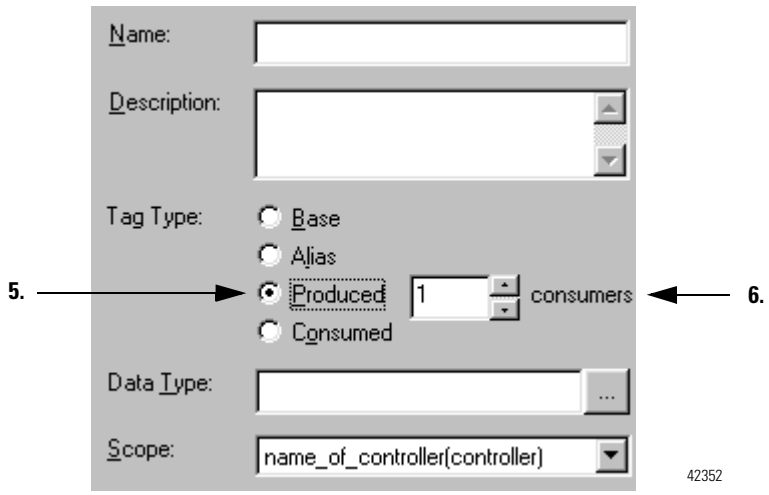
Produce a Tag

1. Open the RSLogix 5000 project that contains the tag that you want to produce.

2. From the *Logic* menu, select *Edit Tags*.

3. From *Scope*, select `name_of_controller`*(controller)*. (Only tags that are controller scope can produce data.)

4. Select the tag that will produce the data and press the *ALT* + *Enter* keys.



5. Select the *Produced* option button.

6. Type or select the number of controllers that will consume (receive) the tag.

7. Click *OK*.

8. Configure a tag in another controller (s) to consume this produced tag. Refer to "Consume a Produced Tag" on page 10-5.

Consume a Produced Tag

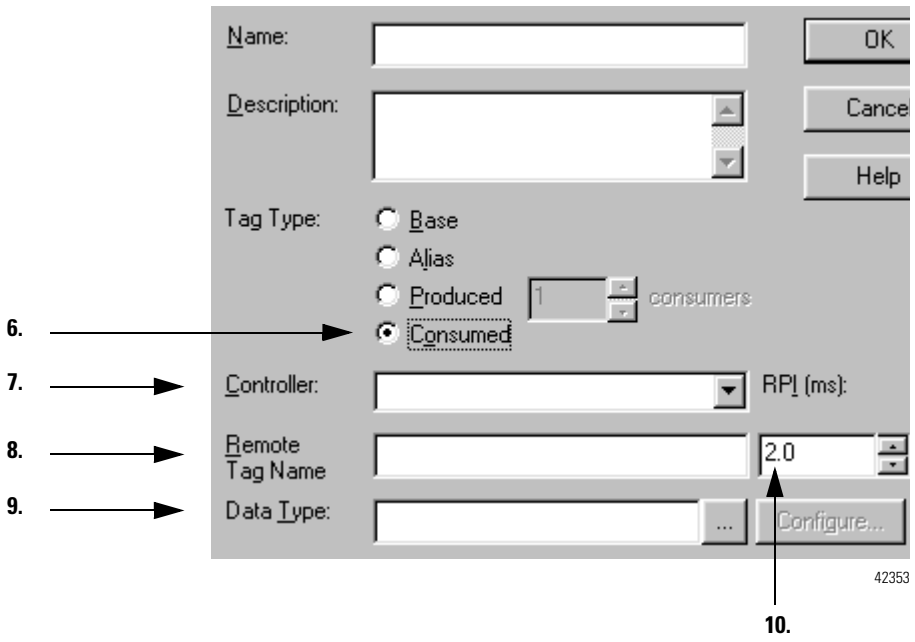
1. Open the RSLogix 5000 project that will consume the **produced tag**.

2. In the controller organizer, *I/O Configuration*, add the controller that contains the produced tag.

3. From the *Logic* menu, select *Edit Tags*.

4. From *Scope*, select *`name_of_controller`(controller)*. (Only tags that are controller scope can consume other tags.)

5. Select the tag in this controller that will consume the produced tag and press the *ALT* + *Enter* keys.



6. Select the *Consumed* option button.

7. Select the controller that contains the produced tag.
8. Type the name of the produced tag.
9. Select the same data type as the produced tag.
10. Type or select the time between updates of the tag:
    - Use the highest value permissible for your application.
    - If the controller consumes the tag over a ControlNet network, use a power of two times the ControlNet **network update time (NUT)**.

      For example, if the NUT is 5 ms, type a rate of 5, 10, 20, 40 ms, etc.
11. Click *OK*.
12. If you share the tag over a ControlNet network, use RSNetWorx for ControlNet software to schedule the network.

IMPORTANT If a consumed-tag connection fails, all of the other tags being consumed from that remote controller stop receiving new data.

Produce Integers for a PLC-5C Controller

1. Open the RSLogx 5000 project.
2. Create a user-defined data type that contains an array of INTs with an even number of elements, such as INT[2]. (When you produce INTs, you must produce two or more.)
3. Create a produced tag and select the user-defined data type from Step 2.
4. Open RSNetWorx™ for ControlNet software.
5. In the ControlNet configuration for the target PLC-5C controller:
   a. Insert a Receive Scheduled Message.
   b. In the Message size, enter the number of integers in the produced tag.
6. In RSNetWorx for ControlNet software, schedule the network.

Produce REALs for a PLC-5C Controller

1. Open the RSLogx 5000 project.
2. How many values do you want to produce?

If you are producing:	Then:
Only one REAL value	Create a produced tag and select the REAL data type.
More than one REAL value	A. Create a user-defined data type that contains an array of REALs. B. Create a produced tag and select the user-defined data type from Step A.

3. Open RSNetWorx for ControlNet software.
4. In the ControlNet configuration for the target PLC-5C controller:
   a. Insert a Receive Scheduled Message.
   b. In the Message size, enter two times the number of REALs in the produced tag. For example, if the produced tag contains 10 REALs, enter 20 for the Message size.

TIP

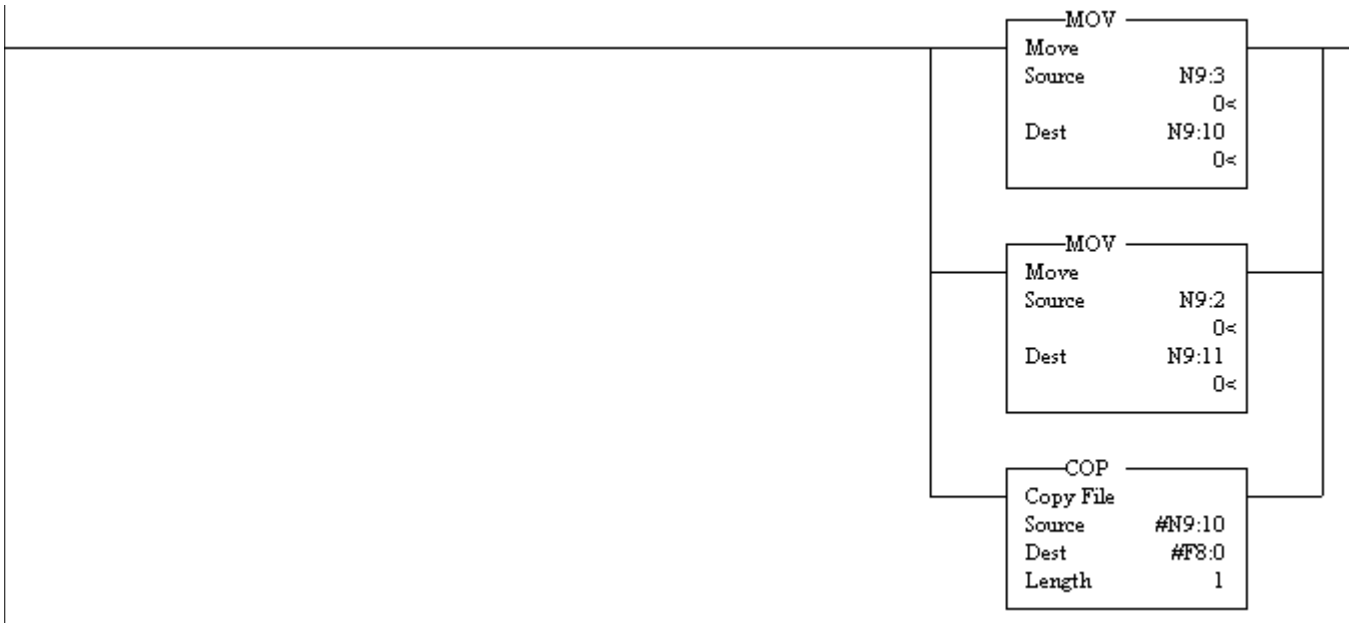
When a PLC-5C controller consumes a tag that is produced by a Logix5000 controller, it stores the data in consecutive 16-bit integers. The PLC-5C stores floating-point data, which requires 32-bits regardless of the type of controller, as follows:

- The first integer contains the upper (left-most) bits of the value.
- The second integer contains the lower (right-most) bits of the value.
- This pattern continues for each floating-point value.

5. In the PLC-5C controller, re-construct the floating point data, as depicted in the following example:

EXAMPLE Re-constructing a floating point value

The two MOV instructions reverse the order of the integers as the integers move to a new location. Because the destination of the COP instruction is a floating-point address, it takes two consecutive integers, for a total of 32 bits, and converts them to a single floating-point value.



6. In RSNetWorx for ControlNet software, schedule the network.

Consume Integers from a PLC-5C Controller

1. Open RSNetWorx for ControlNet software.
2. In the ControlNet configuration of the PLC-5C controller, insert a Send Scheduled Message.
3. Open the RSLogix 5000 project.
4. In the controller organizer, add the PLC-5C controller to the I/O configuration.
5. Create a user-defined data type that contains the following members:

Data type:	Description:
DINT	Status
INT[x], where "x" is the output size of the data from the PLC-5C controller. (If you are consuming only one INT, no dimension is required.)	Data produced by a PLC-5C controller

6. Create a consumed tag with the following properties:

For this tag property:	Type or select:
Tag Type	Consumed
Controller	The PLC-5C that is producing the data
Remote Instance	The message number from the ControlNet configuration of the PLC-5C controller
RPI	A power of two times the NUT of the ControlNet network. For example, if the NUT is 5ms, select an RPI of 5, 10, 20, 40, etc.
Data Type	The user-defined data type that you created in Step 5.

7. In RSNetWorx for ControlNet software, schedule the network.

Adjust for Bandwidth Limitations

When you share a tag over a ControlNet network, the tag must fit within the bandwidth of the network:

- As the number of connections over a ControlNet network increases, several connections, including produced or consumed tags, may need to share a network update time (NUT).
- Since a ControlNet network can only pass 500 bytes in one NUT, the data of each connection must be less then 500 bytes to fit into the NUT.

Depending on the size of your system, you may not have enough bandwidth on your ControlNet network for a tag of 500 bytes. If a tag is too large for your ControlNet network, make one or more of the following adjustments:

- Reduce your network update time (NUT). At a faster NUT, less connections have to share an update slot.
- Increase the requested packet interval (RPI) of your connections. At higher RPIs, connections can take turns sending data during an update slot.
- For a ControlNet bridge module (CNB) in a remote chassis, select the most efficient communication format for that chassis:

Are most of the modules in the chassis non-diagnostic, digital I/O modules?	Then select this communication format for the remote CNB module:
Yes	Rack Optimization
No	None

  The Rack Optimization format uses an additional 8 bytes for each slot in its chassis. Analog modules or modules that are sending or getting diagnostic, fuse, timestamp, or schedule data require direct connections and cannot take advantage of the rack optimized form. Selecting “None” frees up the 8 bytes per slot for other uses, such as produced or consumed tags.

- Separate the tag into two or more smaller tags:
  - Group the data according to similar update rates. For example, you could create one tag for data that is critical and another tag for data that is not as critical.
  - Assign a different RPI to each tag.
- Create logic to transfer the data in smaller sections (packets). Refer to "Produce a Large Array" on page 11-1.

Send a Message

To organize your data for a message, follow these guidelines:

1. For each message, create a tag to control the message:
   - Create the tag at the **controller scope**.
   - Use the MESSAGE data type.

   The following example shows the use of the MESSAGE data type:

EXAMPLE Message to another Logix5000 controller

When *count_send* is on, *count_msg* sends data.

2. For the tags that the messages will use (source or destination tags), create the tags at the controller scope.

3. In the Logix5000 controller, use the **DINT** data type for integers whenever possible:
   – Logix5000 controllers execute more efficiently and use less memory when working with 32-bit integers (DINTs).

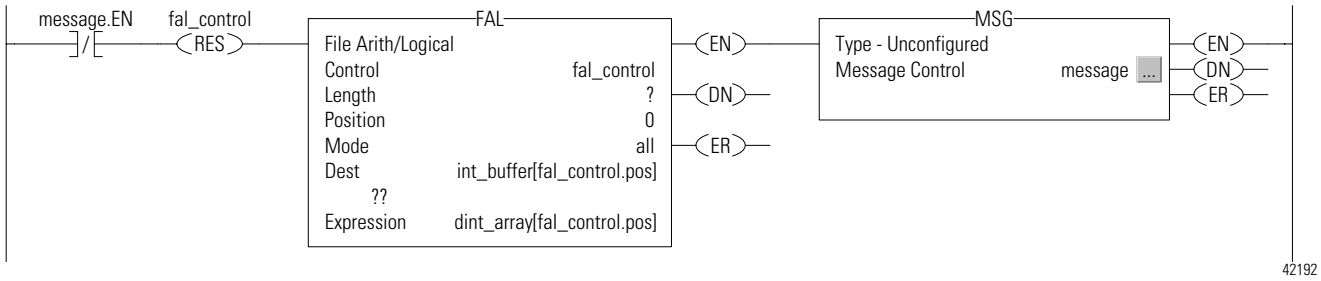
4. If your message is to or from a PLC-5® or SLC 500™ controller *and* it transfers integers (not REALs), use a buffer of **INT**s:
   a. Create a buffer for the data (controller scope) using the *INT[x]* data type.
      where:

      x is the number of integers in the message. (For only one integer, omit *[x].*)

   b. Use the buffer in the message.
   c. Use an FAL instruction to move the data between the buffer and your application.

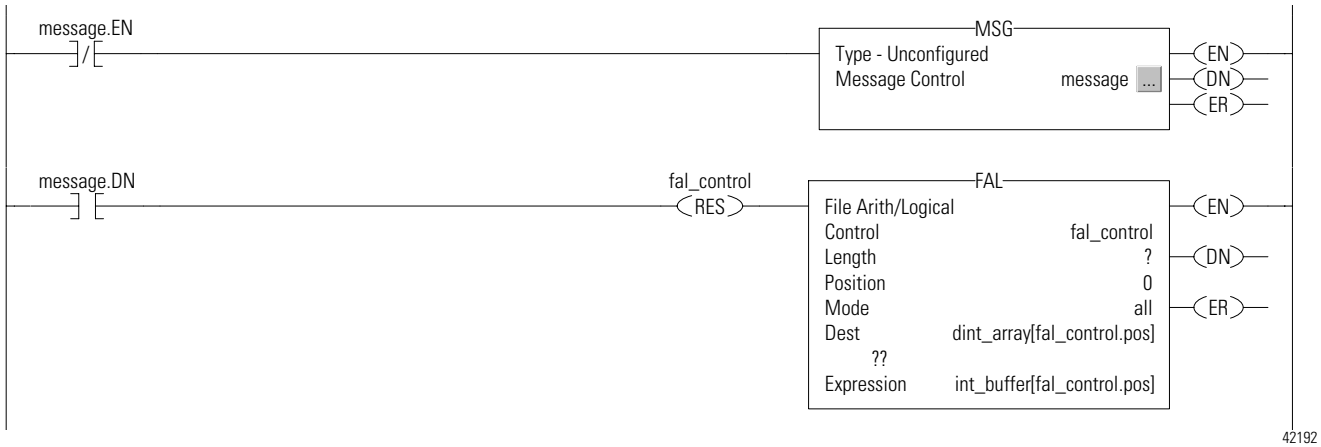
EXAMPLE Write integer values to a PLC-5 controller

Continuously moves the values in *dint_array* to *int_buffer*. This converts the values to 16-bit integers (INTs). Then the message instruction sends *int_buffer* to a PLC-5 controller.



Read integer values from a PLC-5 controller

Continuously reads 16-bit integer values (INTs) from a PLC-5 controller and stores them in *int_buffer*. Then the FAL instruction moves the values to *dint_array*. This converts the values to 32-bit integers (DINTs), for use by other instructions in the project.



Produce a Large Array

When to Use this Procedure

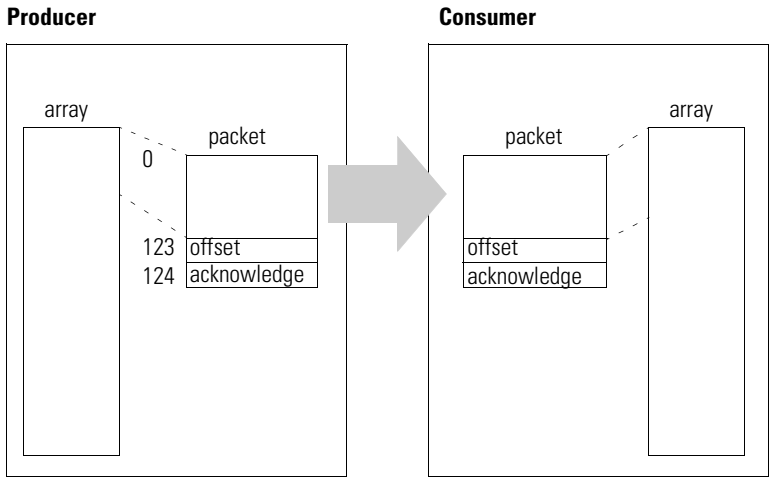
The Logix5000 controller can send as many as 500 bytes of data over a single scheduled connection. This corresponds to 125 DINT or REAL elements of an array. To transfer an array of more than 125 DINTs or REALs, use a produced/consumed tag of 125 elements to create a packet of data. You can then use the packet to send the array piecemeal to another controller.

When you send a large array of data in smaller packets, you must ensure that the transmission of a packet is complete before the data is moved into the destination array, for these reasons.

- Produced data over the ControlLogix backplane is sent in 50 byte segments.
- Data transmission occurs asynchronous to program scan.

The logic that this section includes uses an acknowledge word to make sure that each packet contains new data before the data moves to the destination array. The logic also uses an offset value to indicate the starting element of the packet within the array.

Because of the offset and acknowledge elements, each packet carries 123 elements of data from the array, as depicted below:



In addition, the array must contain an extra 122 elements. In other words, it must be 122 elements greater than the greatest number of elements that you want to transfer:

- These elements serve as a buffer.
- Since each packet contains the same number of elements, the buffer prevents the controller from copying beyond the boundaries of the array.
- Without the buffer, this would occur if the last packet contained fewer than 123 elements of actual data.

Produce a Large Array

1. Open the RSLogix 5000 project that will produce the array.
2. In the Controller Tags folder, create the following tags:

P	Tag Name	Type
	*array*_ack	DINT[2]
✓	*array*_packet	DINT[125]

where:

array is the name for the data that you are sending.

3. Convert *array_ack* to a consumed tag:

For:	Specify:
Controller	name of the controller that is receiving the packet
Remote Tag Name	*array_ack* Both controllers use the same name for this shared data.

Refer to "Consume a Produced Tag" on page 10-5.

4. In either the Controller Tags folder or the tags folder of the program that will contain the logic for the transfer, create the following tags:

Tag Name	Type
array	DINT[x] where x equals the number of elements to transfer plus 122 elements
*array*_offset	DINT
*array*_size	DINT
*array*_transfer_time	DINT
*array*_transfer_time_max	DINT
*array*_transfer_timer	TIMER

where:

array is the name for the data that you are sending.

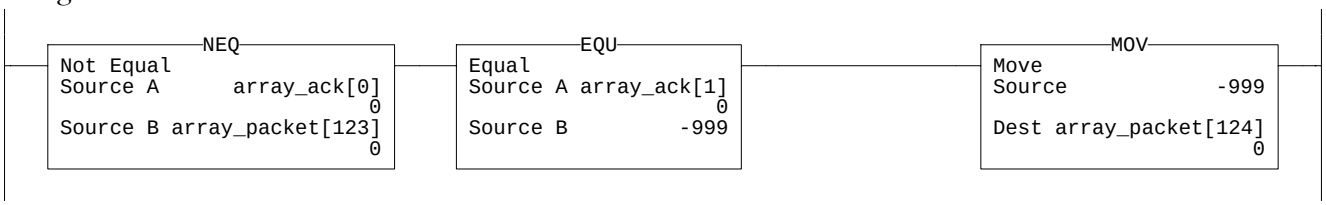
5. In the *array_size* tag, enter the number of elements of real data. (The value of *x* from step *4.* minus the 122 elements of buffer.)

6. Create or open a routine for the logic that will create packets of data.

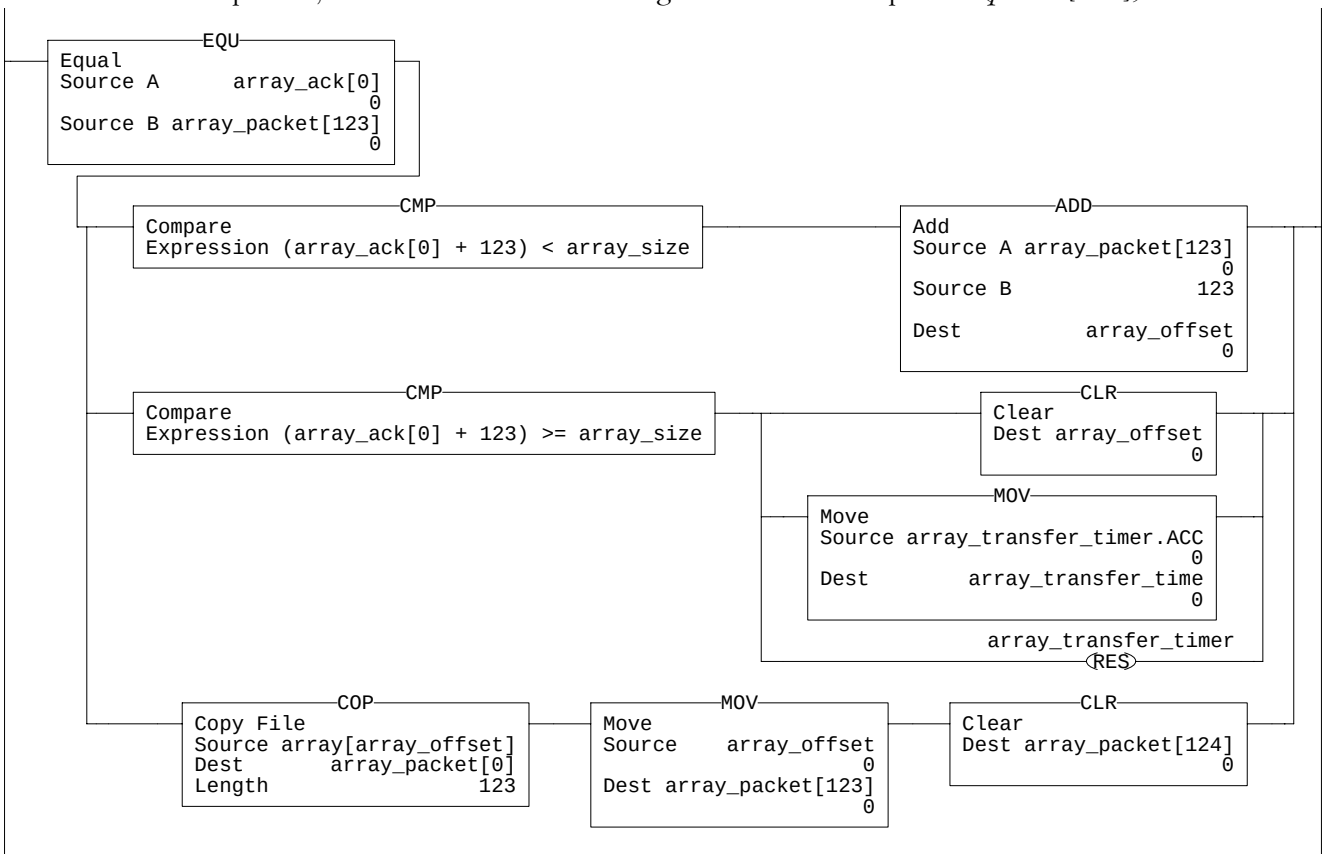
7. Enter the following logic:

Times how long it takes to send the entire array

When the offset value in *array_ack[0]* is not equal to the current offset value but *array_ack[1]* equals -999, the consumer has begun to receive a new packet, so the rung moves -999 into the last element of the packet. The consumer waits until it receives the value -999 before it copies the packet to the array. This guarantees that the consumer has new data.



When the offset value in *array_ack[0]* is equal to the current offset value, the consumer has copied the packet to the array; so the rung checks for more data to transfer. If the offset value plus 123 is less than the size of the array, there is more data to transfer; so the rung increases the offset by 123. Otherwise, there is no more data to transfer; so the rung resets the offset value, logs the transfer time, and resets the timer. In either case, the rung uses the new offset value to create a new packet of data, appends the new offset value to the packet, and clears the acknowledge element of the packet (*packet[124]*).



If the current transfer time is greater than the maximum transfer time, updates the maximum transfer time. This maintains a record of the longest time to transfer data.

42355

8. Open the RSLogix 5000 project that will consume the array.

9. In the Controller Tags folder, create the following tags:

P	Tag Name	Type
✓	*array*_ack	DINT[2]
	*array*_packet	DINT[125]

where:

array is the name for the data that you are sending. Use the same name as in the producing controller (step 2.).

10. Convert *array_packet* to a consumed tag:

For:	Specify:
Controller	name of the controller that is sending the packet
Remote Tag Name	`array`*_packet* Both controllers use the same name for this shared data.

Refer to "Consume a Produced Tag" on page 10-5.

11. In either the Controller Tags folder or the tags folder of the program that will contain the logic for the transfer, create the following tags:

Tag Name	Type
array	DINT[x] where x equals the number of elements to transfer plus 122 elements
*array*_offset	DINT

where:

array is the name for the data that you are sending.

12. Create or open a routine for the logic that will move the data from the packets to the destination array.

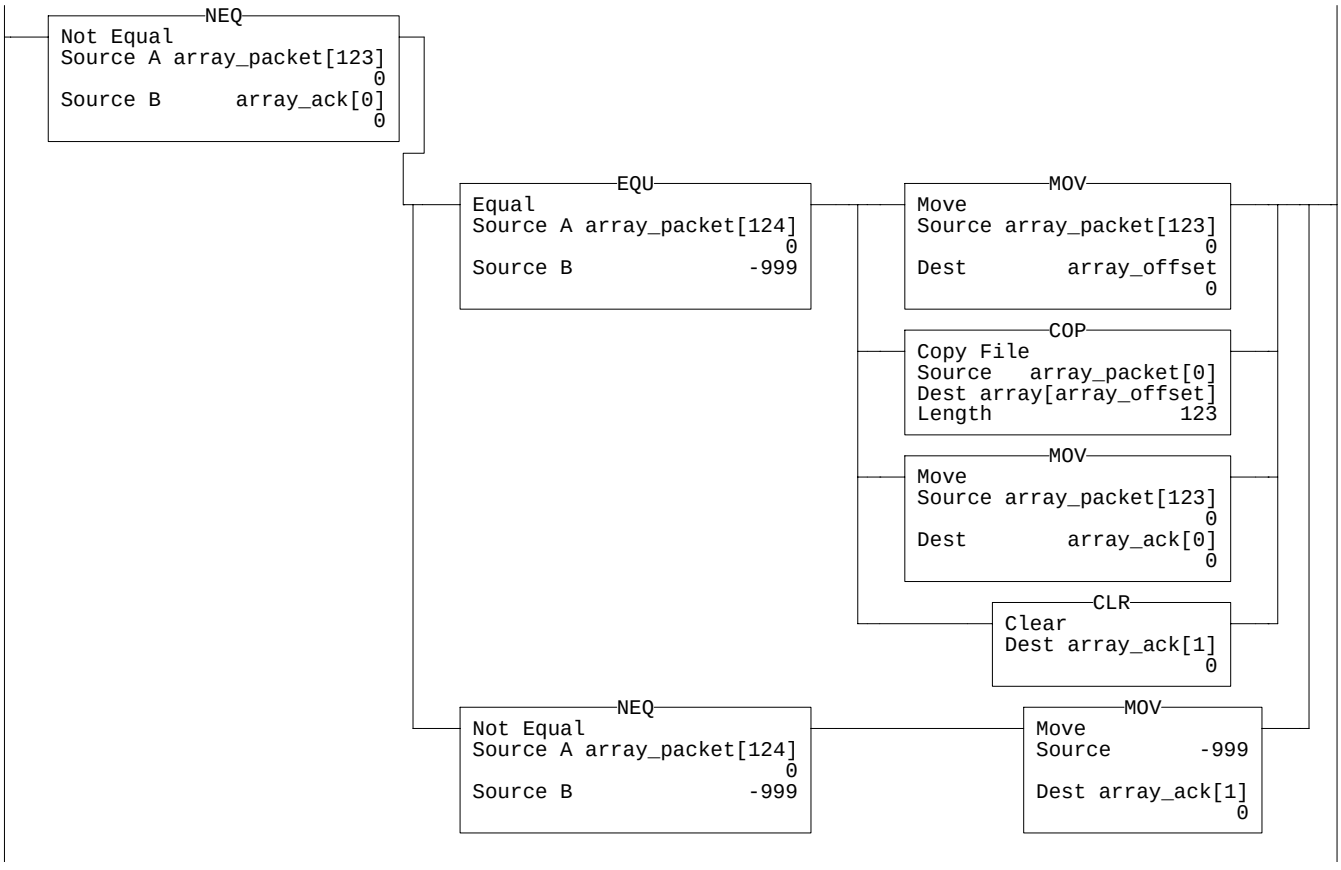
13. Enter the following logic:

When the offset value in *array_packet[123*] is different than the offset value in *array_ack[0]*, the controller has begun to receive a new packet of data; so the rung checks for the value of -999 in the last element of the packet.

If the last element of the packet equals -999, the controller has received an entire packet of new data and begins the copy operation:

- The offset value moves from the packet to array_offset.
- The COP instructions copies the data from the packet to the destination array, starting at the offset value.
- The offset value moves to *array_ack[0]*, which signals that the copy is complete.
- *Array_ack[1]* resets to zero and waits to signal the arrival of a new packet.

If the last element of the packet is not equal -999, the transfer of the packet to the controller may not be complete; so -999 moves to *array_ack[1]*. This signals the producer to return the value of -999 in the last element of the packet to verify the transmission of the packet.



Transferring a large array as smaller packets improves system performance over other methods of transferring the data:

- Fewer connections are used than if you broke the data into multiple arrays and sent each as a produced tag. For example, an array with 5000 elements would take 40 connections (5000/125=40) using individual arrays.

- Faster transmission times are achieved than if you used a message instruction to send the entire array.

  – Messages are unscheduled and are executed only during the "system overhead" portion of the Logix5550 execution. Therefore, messages can take a fairly long time to complete the data transfer.

  – You can improve the transfer time by increasing system overhead time slice, but this diminishes the performance of the continuous task.

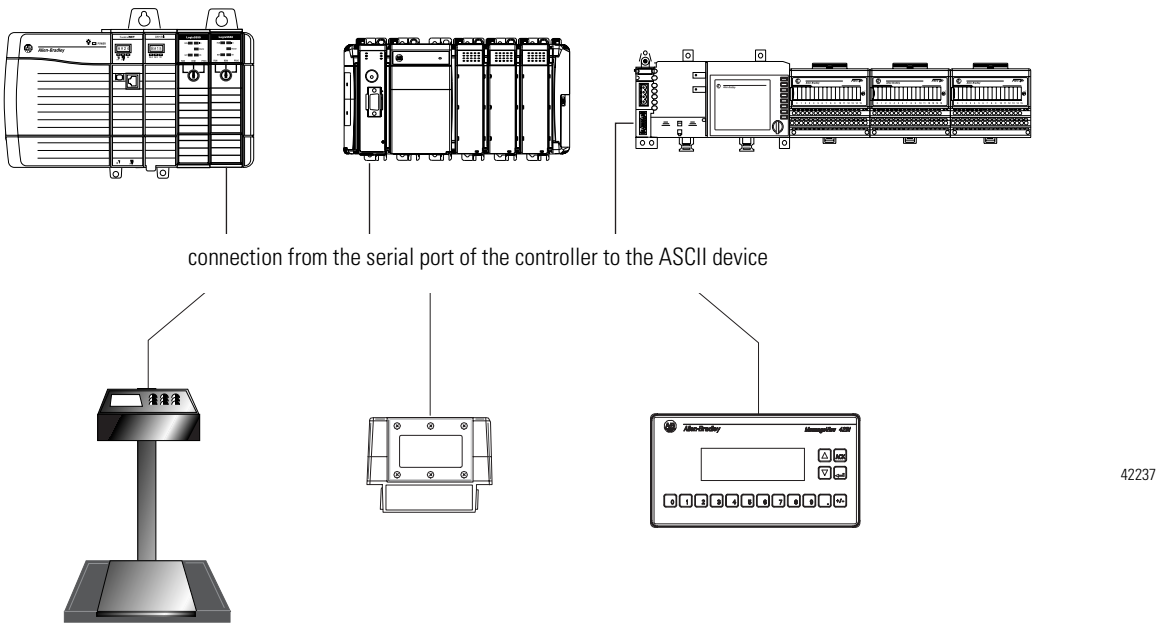
Notes:

Communicate with an ASCII Device

When to Use this Procedure

Use this procedure to exchange ASCII data with a device through the serial port of the controller. For example, you can use the serial port to:

- read ASCII characters from a weigh scale module or bar code reader
- send and receive messages from an ASCII triggered device, such as a MessageView terminal.



How to Use This Procedure

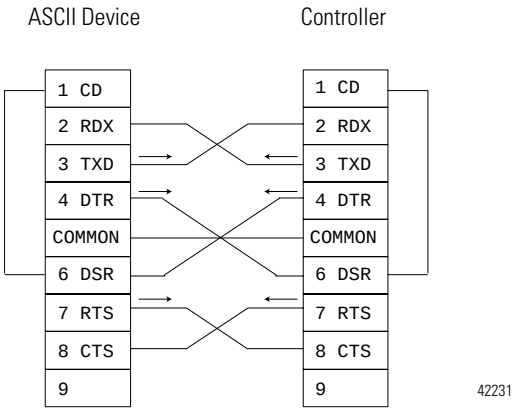
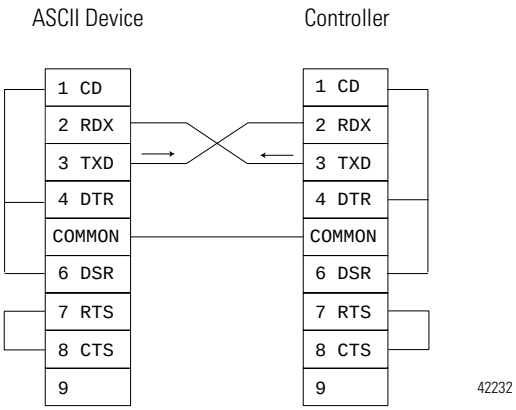
To complete this procedure, do the following tasks:

- Connect the ASCII Device
- Configure the Serial Port
- Configure the User Protocol
- Create String Data Types
- Read Characters from the Device
- Send Characters to the Device

Connect the ASCII Device

1. For the serial port of the ASCII device, determine which pins send signals and which pins receive signals.

2. Connect sending pins to corresponding receiving pins and attach jumpers:

If the communications:	Then wire the connectors as follows:
handshake	
do not handshake	

3. Attach the cable shield to both connectors.

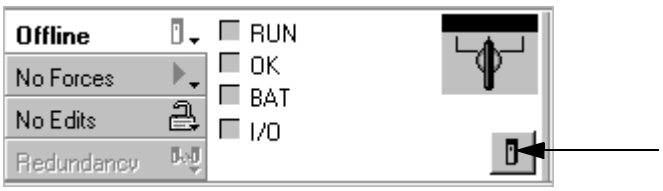
4. Connect the cable to the controller and the ASCII device.

Configure the Serial Port

1. Determine the following communication settings for the ASCII device:

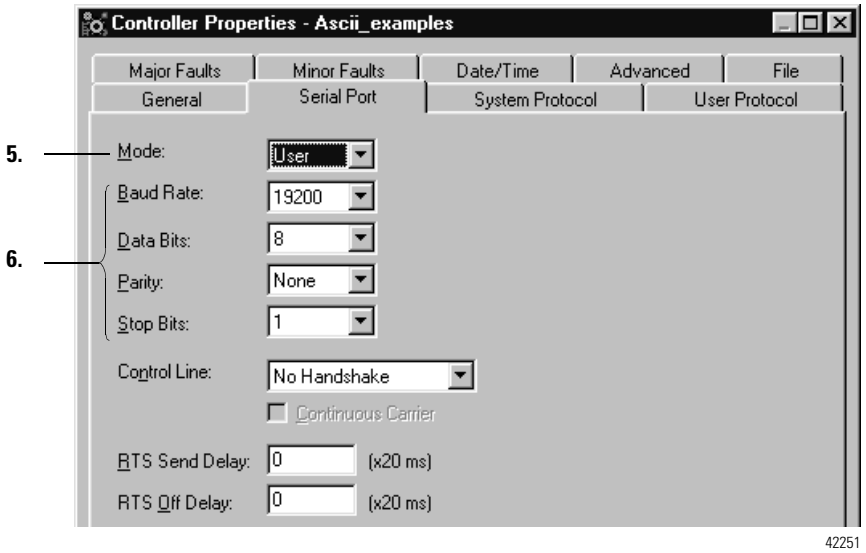
   a. baud rate
   b. data bits
   c. parity
   d. stop bits

2. Open the RSLogix 5000™ project.



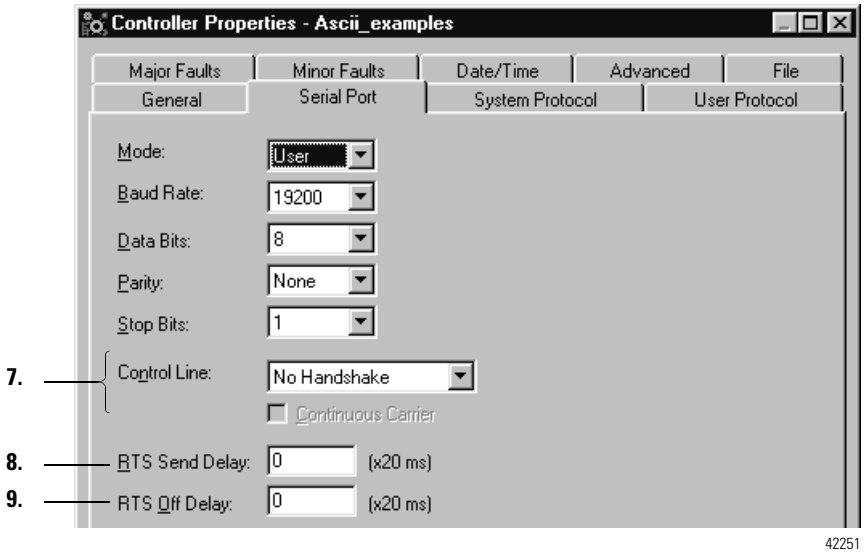
3. On the Online toolbar, click the controller button.

4. Click the *Serial Port* tab.



5. Select *User*.

6. Select the settings for the ASCII device, from step 1.



7. Select the Control Line option:

If:	And:	And this is the:	Select:	Then:
you are *not* using a modem	→		*No Handshaking*	Go to step 10.
you are using a modem	both modems in a point-to-point link are full-duplex	→	*Full Duplex*	
	master modem is full-duplex while slave modem is half-duplex	master controller.	*Full Duplex*	
		slave controller	*Half Duplex*	Select the *Continuous Carrier* check box.
	all modems in the system are half-duplex	→	*Half Duplex*	Clear the *Continuous Carrier* check box (default).

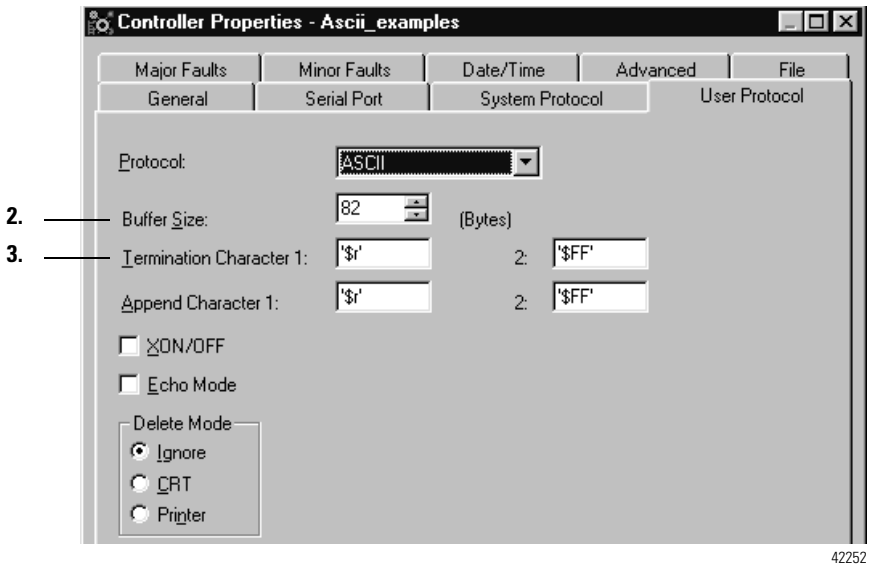
8. Type the amount of delay (20 ms units) between the time that the RTS signal turns on (high) and the time that data is sent. For example, a value of 4 produces an 80 ms delay.

9. Type the amount of delay (20 ms units) between the time that the last character is sent and the time that the RTS signal turns off (low).

10. Click *Apply*.

Configure the User Protocol

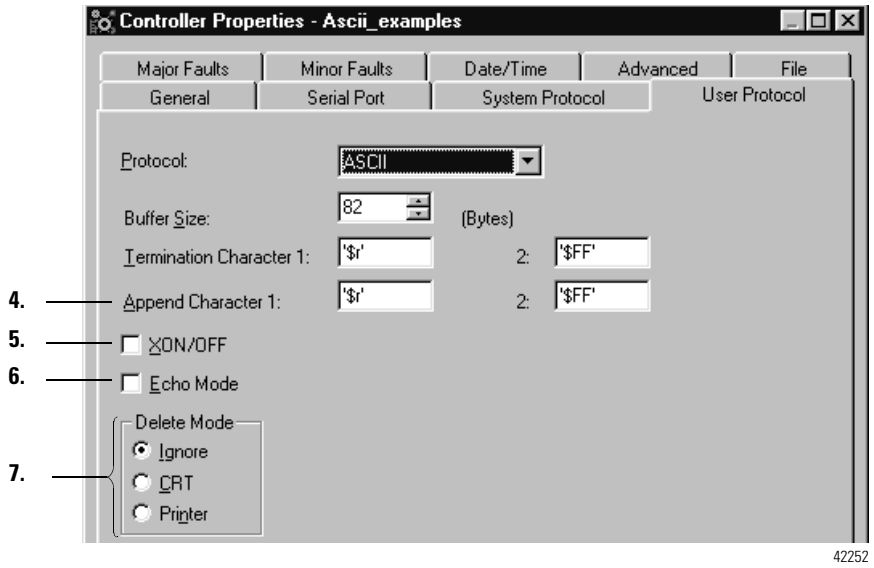
1. Click the *User Protocol* tab.



2. Select or type a number that is greater than or equal to the greatest number of characters in a transmission. (Twice the number of characters is a good guideline.)

3. If you are using ABL or ARL instructions, type the characters that mark the end of the data. For the ASCII code of a character, refer to the back cover of this manual.

If the device sends:	Then:	Notes:
one termination character	A. In the Termination Character 1 text box, type the hexadecimal ASCII code for the first character. B. In the Termination Character 2 text box, type $FF.	For printable characters, such as 1 or A, type the character.
two termination characters	In the Termination Character 1 and 2 text boxes, type the hexadecimal ASCII code for each character.	



4. If you are using the AWA instruction, type the character(s) to append to the data. For the ASCII code of a character, refer to the back cover of this manual.

To append:	Then:	Notes:
one character	A. In the Append Character 1 text box, type the hexadecimal ASCII code for the first character. B. In the Append Character 2 text box, type $FF.	For printable characters, such as 1 or A, type the character.
two characters	In the Append Character 1 and 2 text boxes, type the hexadecimal ASCII code for each character.	

5. If the ASCII device is configured for XON/XOFF flow control, select the *XON/XOFF* check box.

6. If the ASCII device is a CRT or is pre-configured for half duplex transmission, select the *Echo Mode* check box.

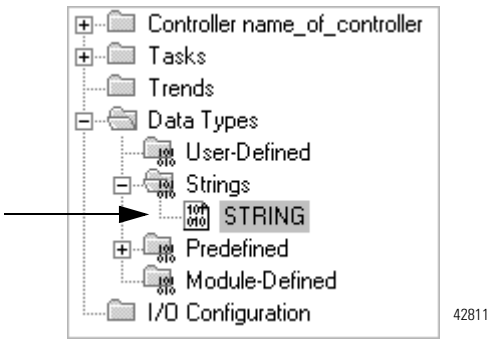
7. Select the Delete Mode:

If the ASCII device is:	Select:	Notes:
CRT	*CRT*	• The DEL character ($7F) and the character that precedes the DEL character are *not* sent to the destination. • If echo mode is selected and an ASCII instruction reads the DEL character, the echo returns three characters: BACKSPACE SPACE BACKSPACE ($08 $20 $08).
printer	*Printer*	• The DEL character ($7F) and the character that precedes the DEL character are *not* sent to the destination. • If echo mode is selected and an ASCII instruction reads the DEL character, the echo returns two characters: / ($2F) followed by the character that was deleted.
None of the above	*Ignore*	The DEL character ($7F) is treated as any other character.

8. Click *OK*.

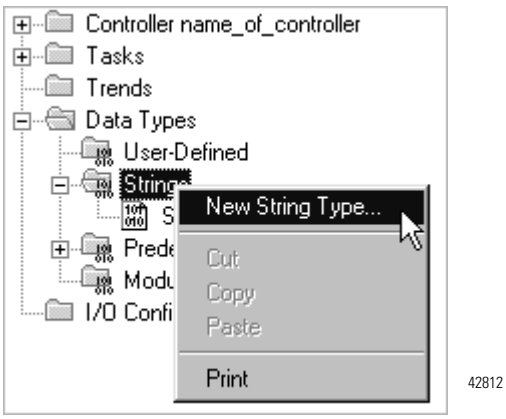
Create String Data Types

You store ASCII characters in tags that use a **string** data type.



You can use the default STRING data type. It stores up to 82 characters.

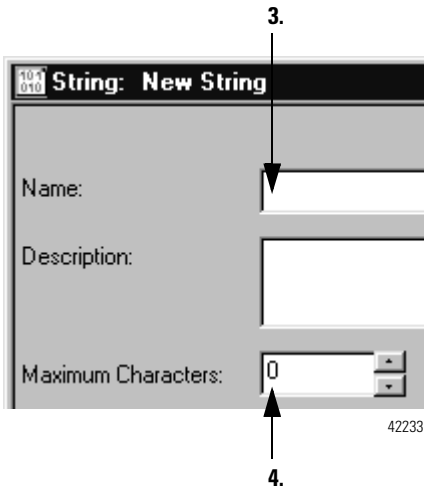
or



You can create a new string data type to store the number of characters that you define.

IMPORTANT Use caution when you create a new string data type. If you later decide to change the size of the string data type, you may lose data in any tags that currently use that data type.

If you:	Then:
make a string data type smaller	• The data is truncated. • The LEN is unchanged.
make a string data type larger	The data and LEN is reset to zero.



1. Do you want to create a new string data type?

If:	Then:
no	Go to Read Characters from the Device on page 12-9.
yes	Go to step 2.

2. In the controller organizer, right-click *Strings* and choose *New String Type…*
3. Type a name for the data type.
4. Type the maximum number characters that this string data type will store.
5. Choose *OK*.

Read Characters from the Device

As a general rule, before you read the buffer use an ACB or ABL instruction to verify that the buffer contains the required characters:

- An ARD or ARL instruction continues to read the buffer until the instruction reads the required characters.
- While an ARD or ARL instruction is reading the buffer, no other ASCII Serial Port instructions, except the ACL, can execute.
- Verifying that the buffer contains the required characters prevents the ARD or ARL from holding up the execution of other ASCII Serial Port instructions while the input device sends its data.

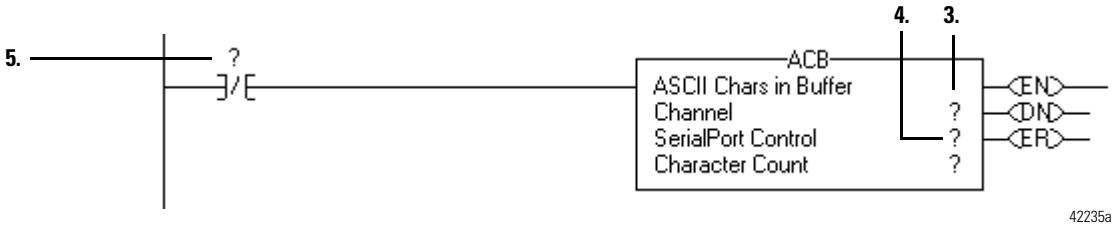
For additional information on ASCII Serial Port instructions, refer to *Logix5000 Controllers General Instruction Set Reference Manual*, publication 1756-RM003.

IMPORTANT If you are not familiar with how to enter ladder logic in an RSLogix 5000 project, first review "Program Routines" on page 4-1.

1. Which type of device are you reading?

If the device is a:	Then:
bar code reader	Go to step 2.
weigh scale that send a fixed number of characters	
message or display terminal	Go to step 14.
weigh scale that send a varying number of characters	

2. Enter the following rung:

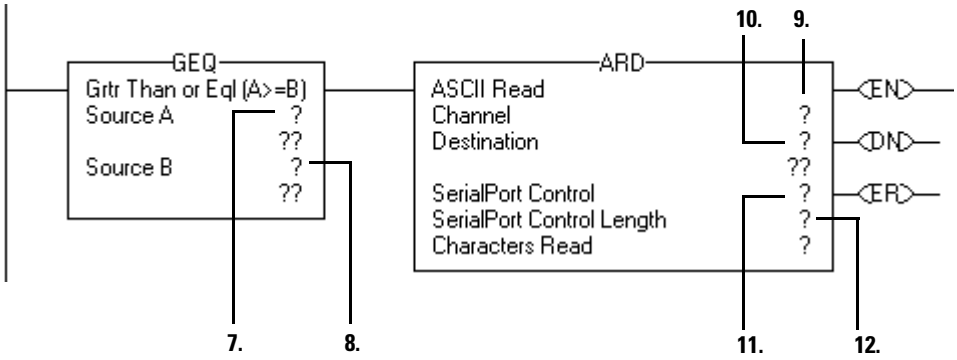


3. Enter 0.

4. Enter a tag name for the ACB instruction and define the data type as SERIAL_PORT_CONTROL.

5. Enter the EN bit of the ACB tag. (The tag from step 4.)

6. Enter the following rung:



7. Enter the POS member of the ACB tag. (The tag from step 4.)

8. Enter the number of characters in the data.

9. Enter 0.

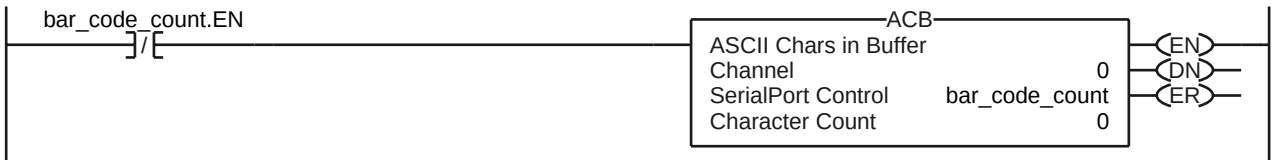
10. Enter a tag name to store the ASCII characters. Define the data type as a **string**.

11. Enter a tag name for the ARD instruction and define the data type as SERIAL_PORT_CONTROL.

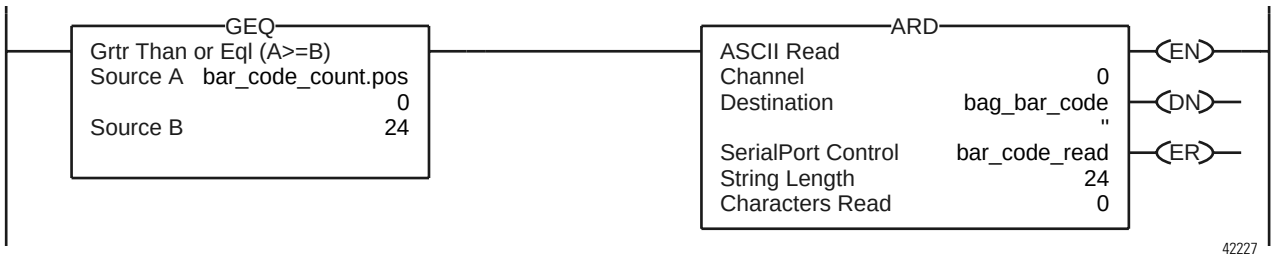
12. Enter the number of characters in the data.

EXAMPLE

A bar code reader sends bar codes to the serial port (channel 0) of the controller. Each bar code contains 24 characters. To determine when the controller receives a bar code, the ACB instruction continuously counts the characters in the buffer.



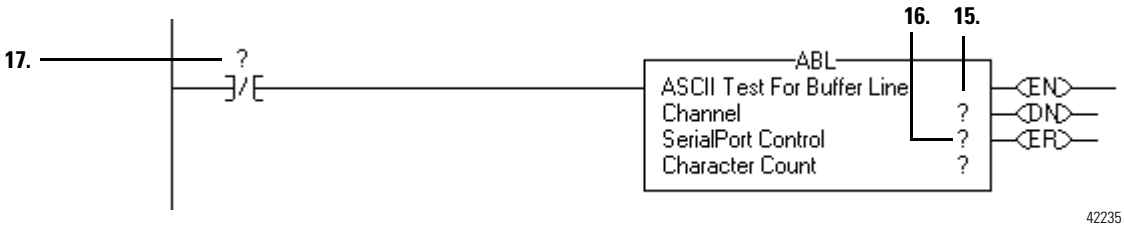
When the buffer contains at least 24 characters, the controller has received a bar code. The ARD instruction moves the bar code to the *bag_bar_code* tag.



13. Do you want to send data to the device?

If:	Then:
yes	Go to Send Characters to the Device on page 12-14.
no	Stop. You are done with this procedure. To use the data, go to "Process ASCII Characters" on page 13-1.

14. Enter the following rung:

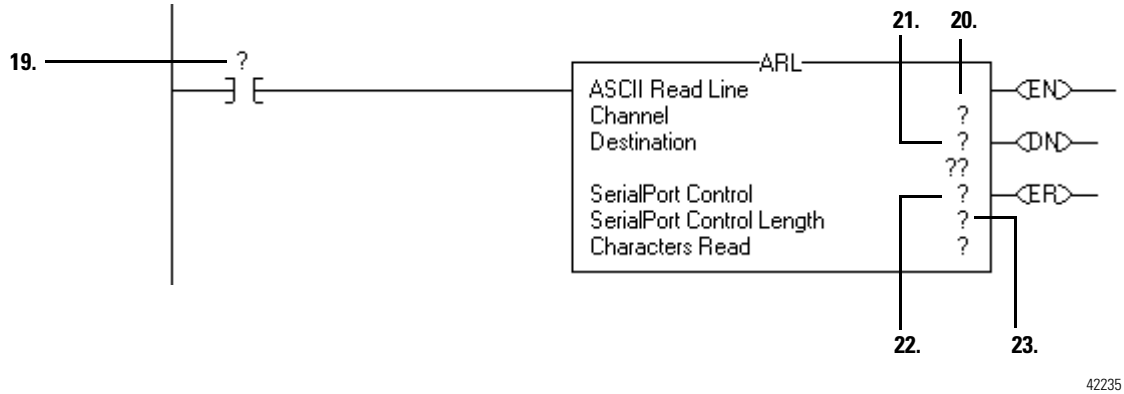


15. Enter 0.

16. Enter a tag name for the ABL instruction and define the data type as SERIAL_PORT_CONTROL.

17. Enter the EN bit of the ABL tag. (The tag from step 16.)

18. Enter the following rung:



19. Enter the FD bit of the ABL tag. (The tag from step 16.)

20. Enter 0.

21. Enter a tag name to store the ASCII characters. Define the data type as a **string**.

22. Enter a tag name for the ARL instruction and define the data type as SERIAL_PORT_CONTROL.

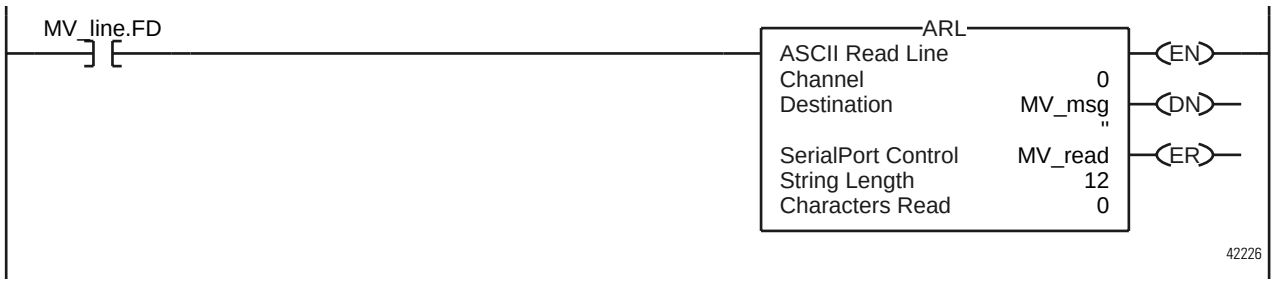
23. Enter the maximum number of characters to read if no termination character (s) is found.

EXAMPLE

Continuously tests the buffer for a message from the MessageView terminal.

- Since each message ends in a carriage return ($0D), the carriage return is configured as the termination character in the Controller Properties dialog box, User Protocol tab.
- When the ABL finds a carriage return, its sets the FD bit.

When the ABL instruction finds the carriage return (MV_line.FD is set), the controller removes the characters from the buffer, up to and including the carriage return, and places them in the *MV_msg* tag.



24. Do you want to send data to the device?

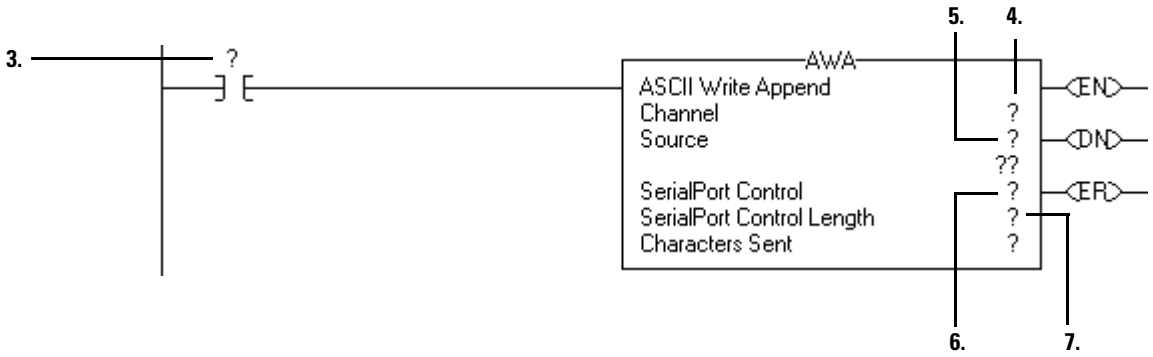
If:	Then:
yes	Go to Send Characters to the Device on page 12-14.
no	Stop. You are done with this procedure. To use the data, go to "Process ASCII Characters" on page 13-1.

Send Characters to the Device

1. Determine where to start:

If you:	And you:	Then:
always send the same number of characters	want to automatically append one or two characters to the end of the data	Go to step 2.
	do not want to append characters	Go to step 9.
send different numbers of characters	want to automatically append one or two characters to the end of the data	Go to step 16.
	do not want to append characters	Go to step 24.

2. Enter the following rung:



42236a

3. Enter the input condition (s) that determines when the characters are to be sent:
   - You can use any type of input instruction.
   - The instruction must change from false to true each time the characters are to be sent.

4. Enter 0.

5. Enter the tag name that stores the ASCII characters. Define the data type as a **string**.

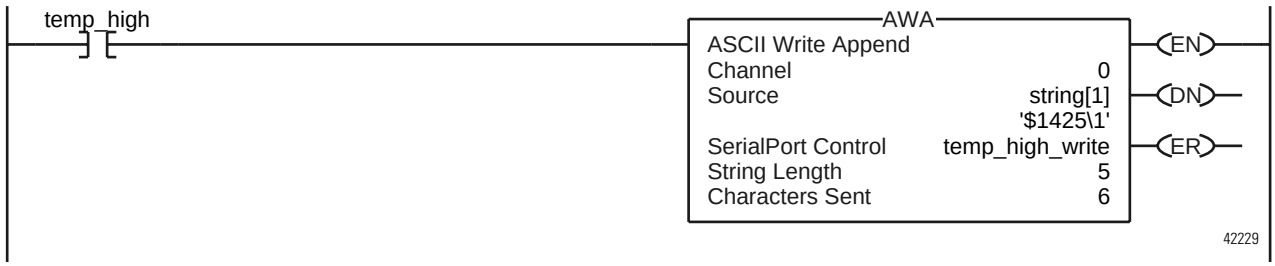
6. Enter a tag name for the AWA instruction and define the data type as SERIAL_PORT_CONTROL.

7. Enter the number of characters to send. Omit the characters that are appended by the instruction.

EXAMPLE

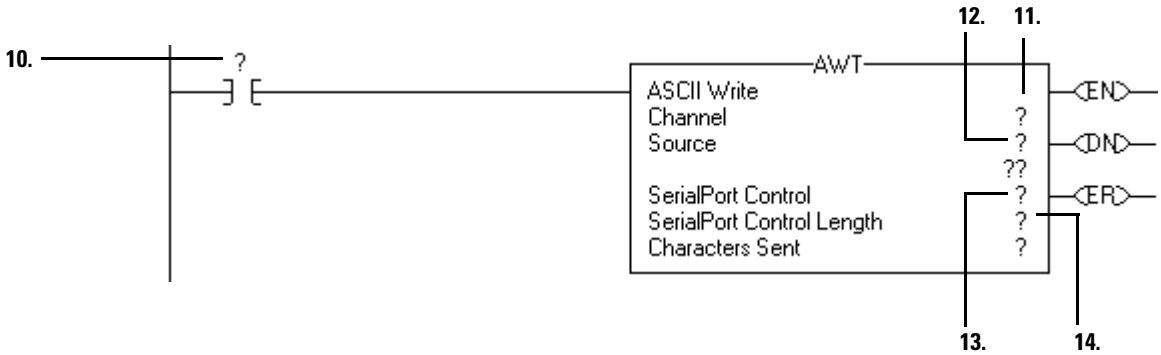
When the temperature exceeds the high limit (*temp_high* is on), the AWA instruction sends five characters from the *string[1]* tag to a MessageView terminal.

- The *$14* counts as one character. It is the hex code for the Ctrl-T character.
- The instruction also sends (appends) the characters defined in the user protocol. In this example, the AWA instruction sends a carriage return ($0D), which marks the end of the message.



8. Go to Enter ASCII Characters on page 12-21.

9. Enter the following rung:



10. Enter the input condition (s) that determines when the characters are to be sent:

- You can use any type of input instruction.
- The instruction must change from false to true each time the characters are to be sent.

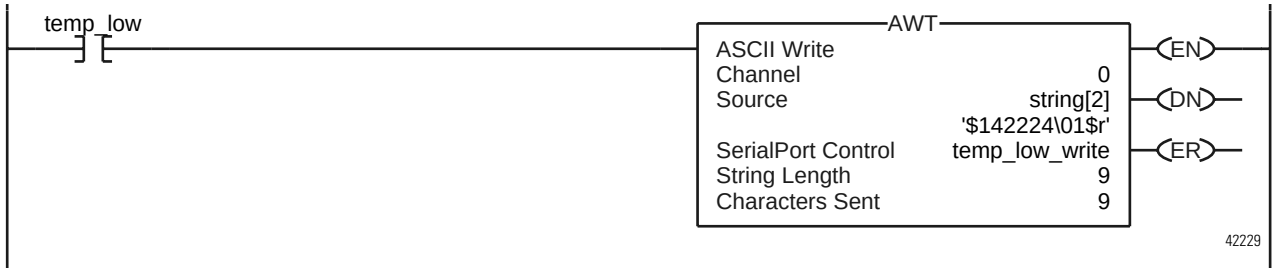
11. Enter 0.

12. Enter the tag name that stores the ASCII characters. Define the data type as a **string**.

13. Enter a tag name for the AWT instruction and define the data type as SERIAL_PORT_CONTROL.

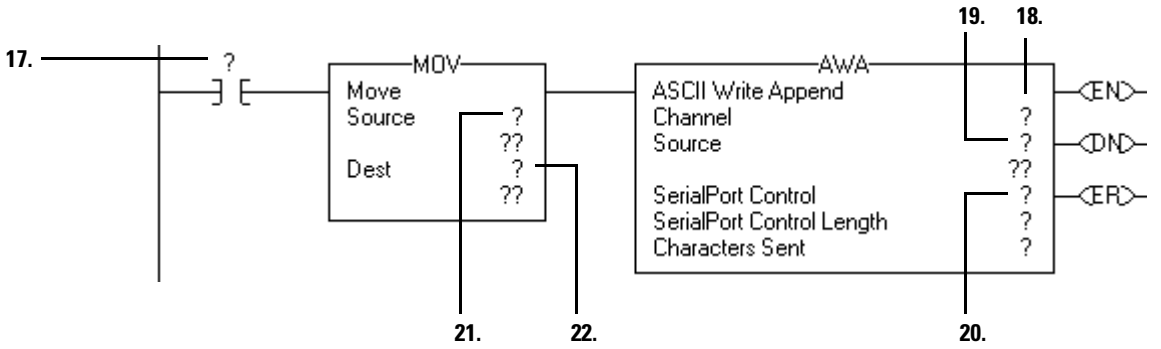
14. Enter the number of characters to send.

EXAMPLE When the temperature reaches the low limit (*temp_low* is on), the AWT instruction sends nine characters from the *string[2]* tag to a MessageView terminal. (The *$14* counts as one character. It is the hex code for the Ctrl-T character.)



15. Go to Enter ASCII Characters on page 12-21.

16. Enter the following rung:



42236c

17. Enter the input condition (s) that determines when the characters are to be sent:

- You can use any type of input instruction.
- The instruction must change from false to true each time the characters are to be sent.

18. Enter 0.

19. Enter the tag name that stores the ASCII characters. Define the data type as a **string**.

20. Enter a tag name for the AWA instruction and define the data type as SERIAL_PORT_CONTROL.

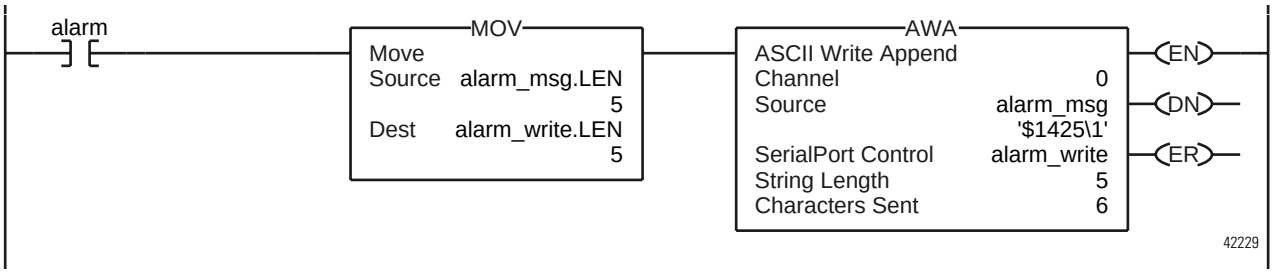
21. Enter the LEN member of the Source tag. (The tag from step 19.)

22. Enter the LEN member of the AWA instruction. (The tag from step 20.)

EXAMPLE

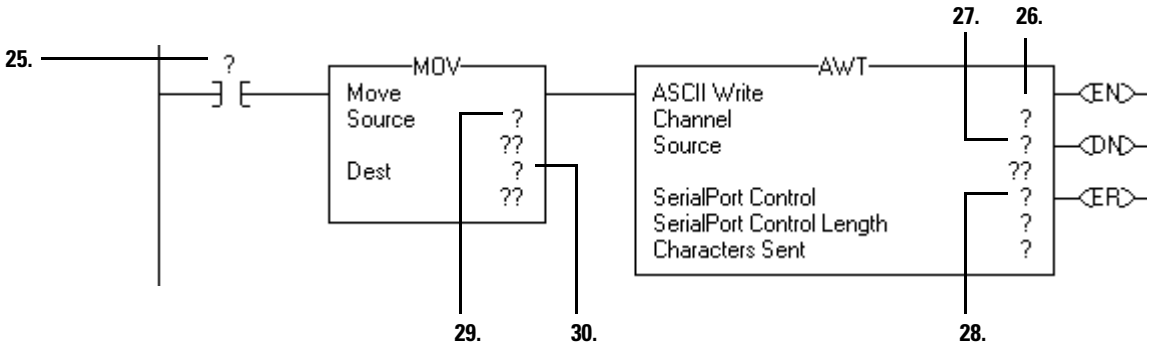
When *alarm* is on, the AWA instruction sends the characters in *alarm_msg* and appends a termination character.

- Because the number of characters in *alarm_msg* varies, the rung first moves the length of *alarm_msg* (*alarm_msg*.LEN) to the length of the AWA instruction (*alarm_write*.LEN).
- In *alarm_msg*, the *$14* counts as one character. It is the hex code for the Ctrl-T character.



23. Go to Enter ASCII Characters on page 12-21.

24. Enter the following rung:



42236d

25. Enter the input condition (s) that determines when the characters are to be sent:

- You can use any type of input instruction.
- The instruction must change from false to true each time the characters are to be sent.

26. Enter 0.

27. Enter the tag name that stores the ASCII characters. Define the data type as a **string**.

28. Enter a tag name for the AWT instruction and define the data type as SERIAL_PORT_CONTROL.

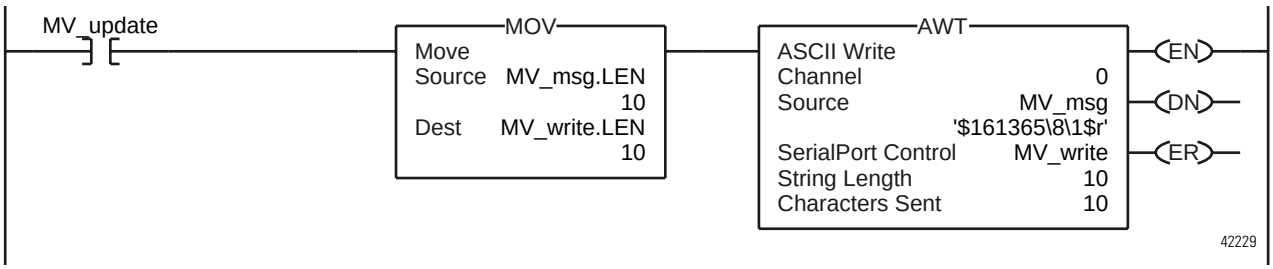
29. Enter the LEN member of the Source tag. (The tag from step 27.)

30. Enter the LEN member of the AWT instruction. (The tag from step 28.)

EXAMPLE

When *MV_update* is on, the AWT instruction sends the characters in *MV_msg*.

- Because the number of characters in *MV_msg* varies, the rung first moves the length of *MV_msg* (*MV_msg*.LEN) to the length of the AWT instruction (*MV_write*.LEN).
- In *MV_msg*, the *$16* counts as one character. It is the hex code for the Ctrl-V character.



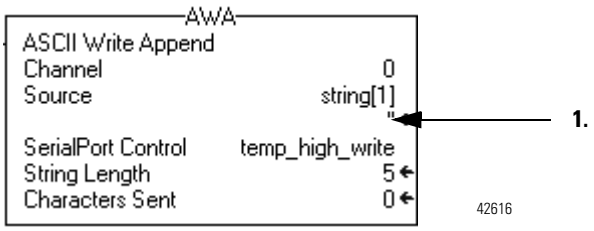
31. Go to Enter ASCII Characters on page 12-21.

Enter ASCII Characters

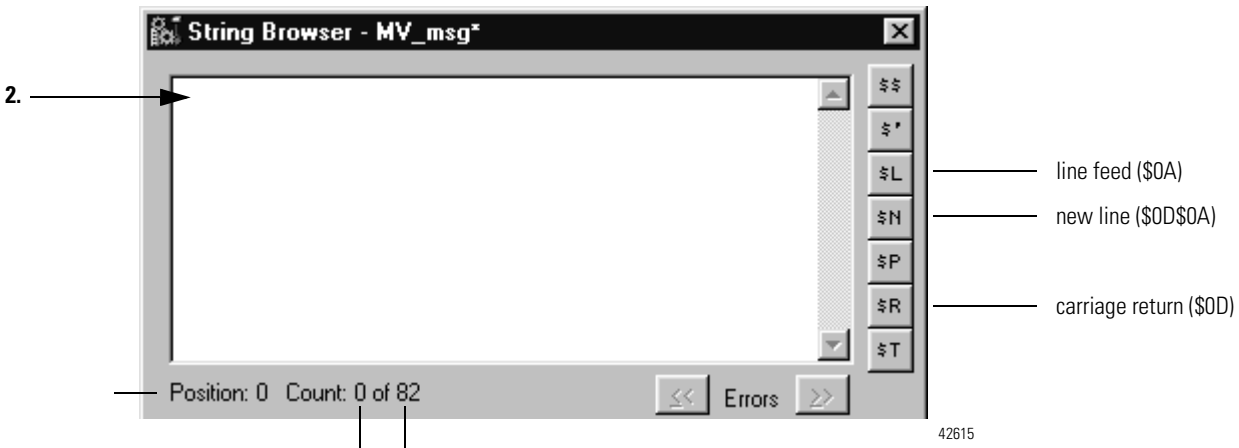
Determine if you must complete this step:

If:	Then:
You want logic to create the string.	Go to "Process ASCII Characters" on page 12-1.
You want to enter the characters.	Go to step 1.

IMPORTANT This String Browser window shows the characters up to the value of the LEN member of the string tag. The string tag may contain additional data, which the String Browser window does not show.



1. Double-click the value area of the Source.



2. Type the characters for the string.

3. Choose *OK*.

Notes:

Process ASCII Characters

When to Use this Procedure

Use this procedure to:

- interpret a bar code and take action based on the bar code
- use a weight from a weigh scale when the weight is sent as ASCII characters
- decode a message from an ASCII triggered device, such as an operator terminal
- build a string for an ASCII triggered device using variables from your application

How to Use this Procedure

IMPORTANT If you are not familiar with how to enter ladder logic in an RSLogix 5000 project, first review "Program Routines" on page 4-1.

Depending on your application, you may not need to do all the tasks in this procedure. Use the following table to determine where to start:

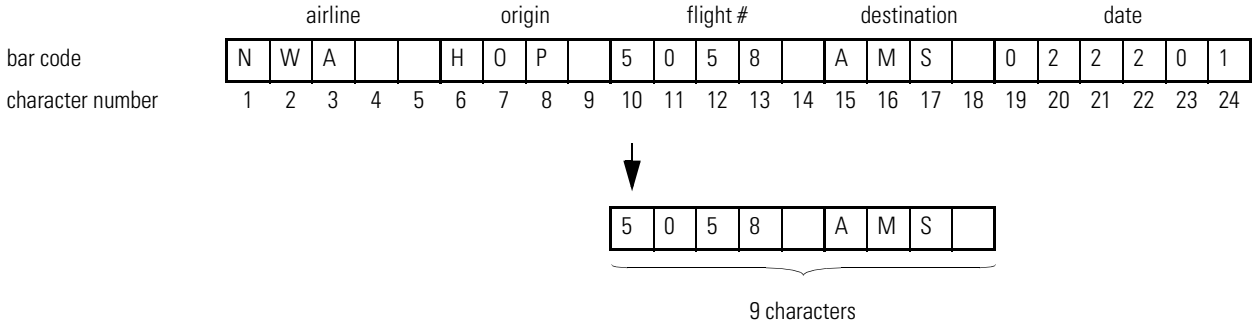
If you want to:	Then go to:	On page:
initiate an action based on a bar code	Extract a Part of a Bar Code	13-2
use a weight from a weigh scale	Convert a Value	13-6
decode a message from an operator terminal	Decode an ASCII Message	13-8
create a string to send to an operator terminal	Build a String	13-12

For additional information on ASCII-related instructions, refer to *Logix5000 Controllers General Instruction Set Reference Manual*, publication 1756-RM003.

Extract a Part of a Bar Code

Use the following steps to extract a part of a bar code so you can take action based on its value.

For example, a bar code may contain information about a bag on a conveyor at an airport. To check the flight number and destination of the bag, you extract characters 10 - 18.


airline
origin
flight #
destination
date
bar code
N W A H O P 5 0 5 8 A M S 0 2 2 2 0 1
character number
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
5 0 5 8 A M S
9 characters

Steps:

1. Enter the following rung:

2. Enter the EM bit of the ARD instruction that read the bar code.

3. Enter the **string** tag that contains the bar code.

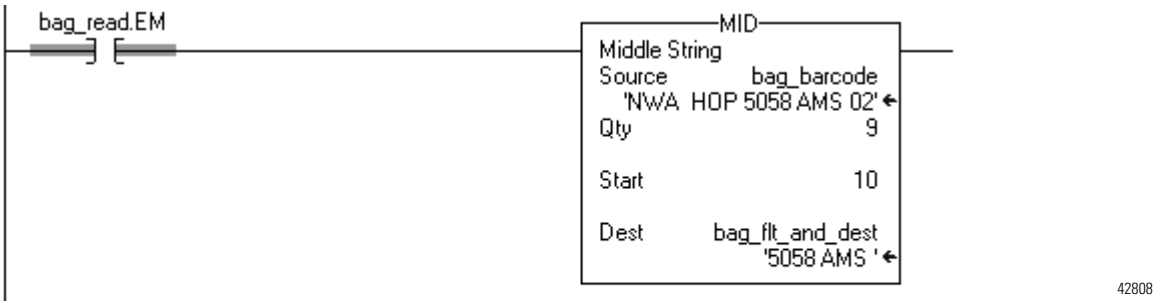
4. Enter the number of characters in the part of the bar code that you want to check.

5. Enter the position of the first character in the part of the bar code that you want to check.

6. Enter a tag name to store the part of the bar code that you want to check. Define the data type as a string.

EXAMPLE

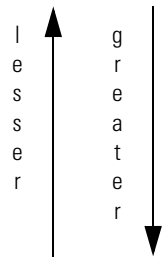
In the baggage handling conveyor of an airport, each bag gets a bar code. Characters 10 - 18 of the bar code are the flight number and destination airport of the bag. After the bar code is read (*bag_read*.EM is on) the MID instruction copies the flight number and destination airport to the *bag_flt_and_dest* tag.



Check the Bar Code Characters

In this task, you use a compare instruction (EQU, GEQ, GRT, LEQ, LES, NEQ) to check for specific characters.

- The hexadecimal values of the characters determine if one string is less than or greater than another string.
- When the two strings are sorted as in a telephone directory, the order of the strings determines which one is greater.



ASCII Characters	Hex Codes
1ab	$31$61$62
1b	$31$62
A	$41
AB	$41$42
B	$42
a	$61
ab	$61$62

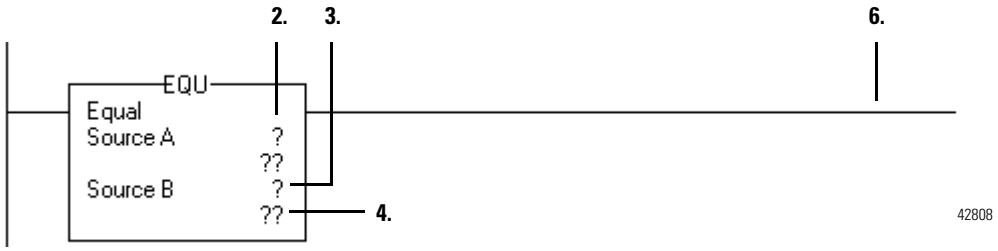
AB < B

a > B

Steps:

1. Enter a rung and a compare instruction:

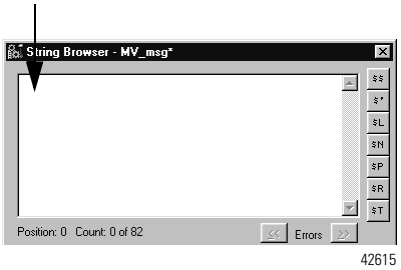
To see if the string is:	Enter this instruction:
equal to specific characters	EQU
not equal to specific characters	NEQ
greater than specific characters	GRT
equal to or greater than specific characters	GEQ
less than specific characters	LES
equal to or less than specific characters	LEQ



2. Enter the tag that stores the part of the bar code that you want to check. (The Destination from Extract a Part of a Bar Code, step 6.)

3. Enter a tag name to store the characters that you want to test against. Define the data type as a **string**.

4. Double-click value area of Source B.

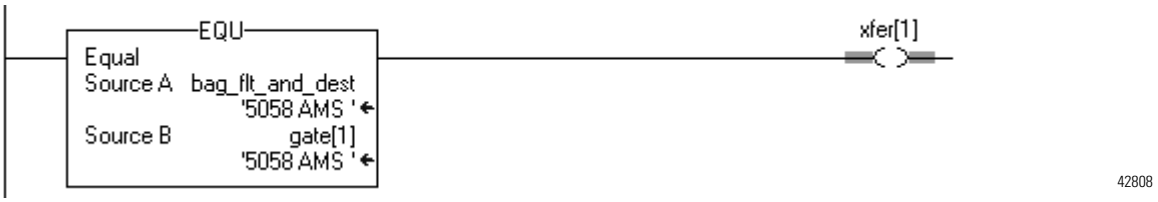


5. Type the ASCII characters to test against and choose *OK*.

6. Enter the required output.

EXAMPLE

When *bag_flt_and_dest* is equal to *gate[1]*, *xfer[1]* turns on. This routes the bag to the required gate.



7. Do you want to check another part of the bar code?

If:	Then:
yes	Go to Extract a Part of a Bar Code on page 13-2.
no	Stop. You are done with this procedure.

Convert a Value

Use the following steps to convert the ASCII representation of a value to an DINT or REAL value that you can use in your application.

- The STOD and STOR instructions skip any initial control or non-numeric characters (except the minus sign in front of a number).
- If the string contains multiple groups of numbers that are separated by delimiters (e.g., /), the STOD and STOR instructions convert only the first group of numbers.

Steps:

1. Which type of number is the value?

If:	Then:
floating-point	Go to step 2.
integer	Go to step 7.

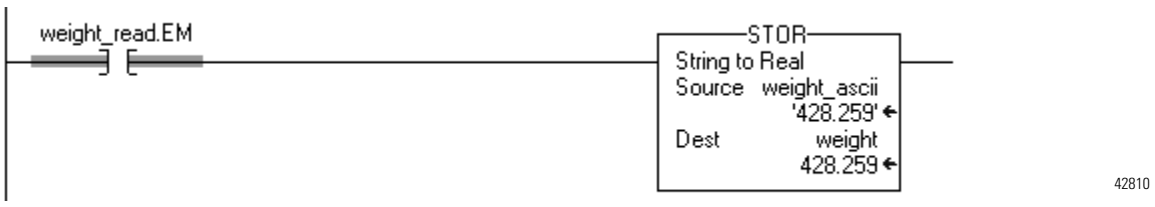
2. Enter the following rung:

3. Enter the EM bit of the ARD or ARL instruction that read the value.

4. Enter the **string** tag that contains the value.

5. Enter a tag name to store the value for use in your application. Define the data type as REAL.

EXAMPLE After reading the weight from the scale (*weight_read*.EM is on) the STOR instruction converts the numeric characters in *weight_ascii* to a REAL value and stores the result in *weight*.



6. Go to step 11.

7. Enter the following rung:

8. Enter the EM bit of the ARD or ARL instruction that read the value.

9. Enter the **string** tag that contains the value.

10. Enter a tag name to store the value for use in your application. Define the data type as DINT.

EXAMPLE

When *MV_read.EM* is on, the STOD instruction converts the first set of numeric characters in *MV_msg* to an integer value. The instruction skips the initial control character ($06) and stops at the delimiter (\ ).

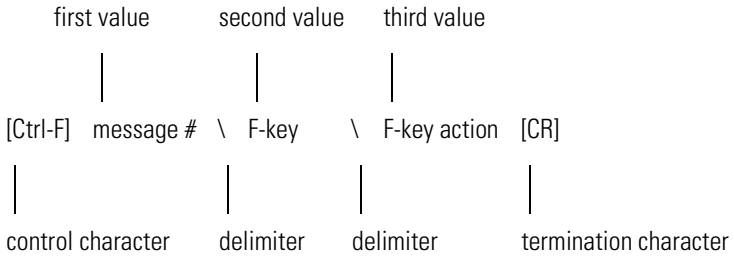
11. Does this string have another value that you want to use?

If:	Then:
yes	Go to Decode an ASCII Message on page 13-8.
no	Stop. You are done with this procedure.

Decode an ASCII Message

Use the following steps to extract and convert a value from an ASCII message that contains multiple values.

For example, a message may look like this:



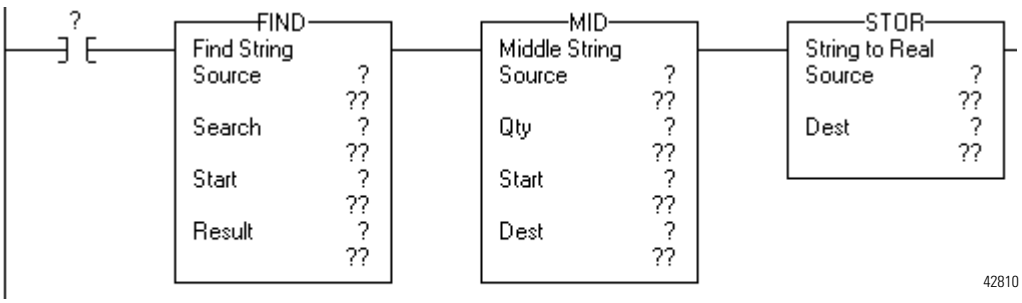
1. Determine where to start:

If the:	And:	Then:
string has more than one value	This is the first value.	Go to Convert a Value on page 13-6.
	This is *not* the value.	Go to step 2.
string has only one value	⟶	Go to Convert a Value on page 13-6.

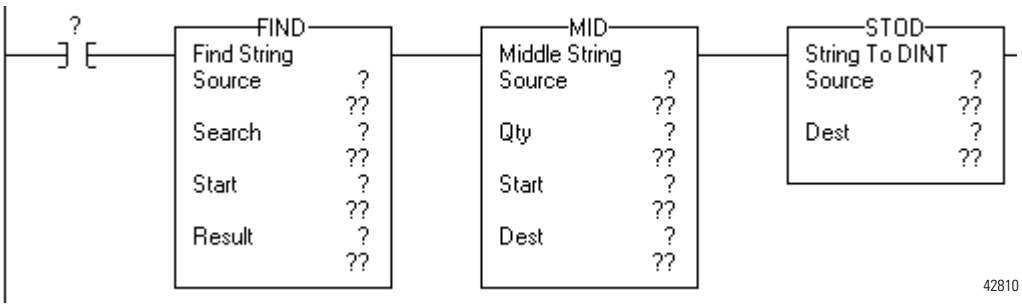
2. Which type of number is the value?

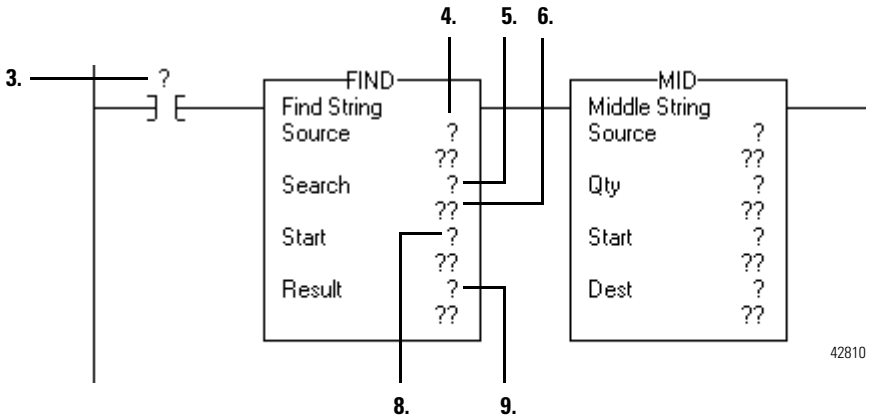
If:	Then:
floating-point	Enter Rung A: Find and Convert a Floating-Point Value
integer	Enter Rung B: Find and Convert an Integer Value

Rung A: Find and Convert a Floating-Point Value



Rung B: Find and Convert an Integer Value





3. Enter the EM bit of the ARL instruction that read the value.

4. Enter the **string** tag that contains the value.

5. Enter a tag name to store the delimiter that marks the beginning of the value. Define the data type as a string.

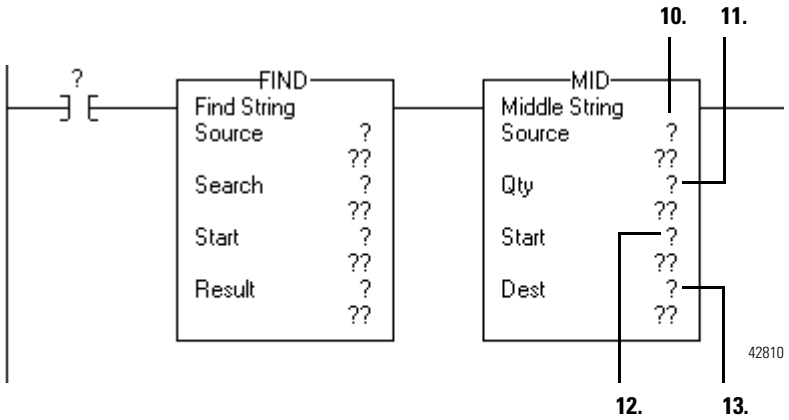
6. Double-click the value area of Search.

7. Type the delimiter and choose *OK*.

8. Enter the position in the string to start the search.

- Initially, you can use 0 to find the first delimiter.
- To decode additional data, increase this value to search for the next delimiter.

9. Enter a tag name to store the location of the delimiter. Define the data type as a DINT.



10. Enter the string tag that contains the value.

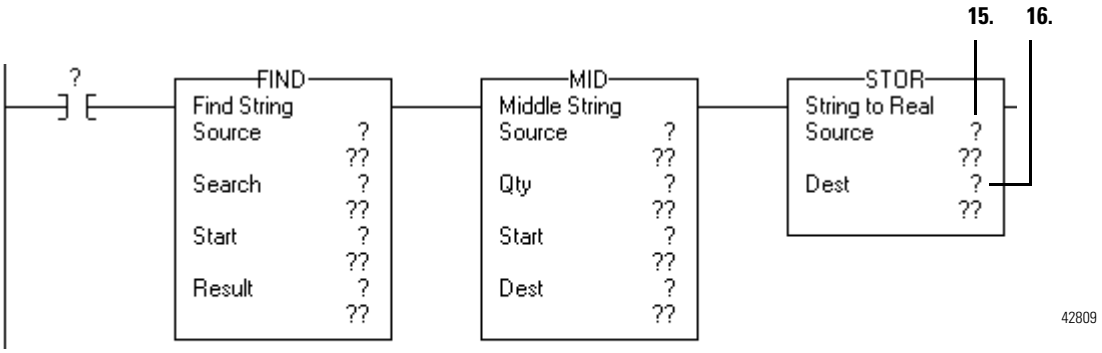
11. Enter the maximum number of characters that this value can contain.

12. Enter the tag that stores the position of the delimiter. (The tag from step 9.)

13. Enter a tag name to store this value. Define the data type as a string.

14. Which type of conversion instruction did you use?

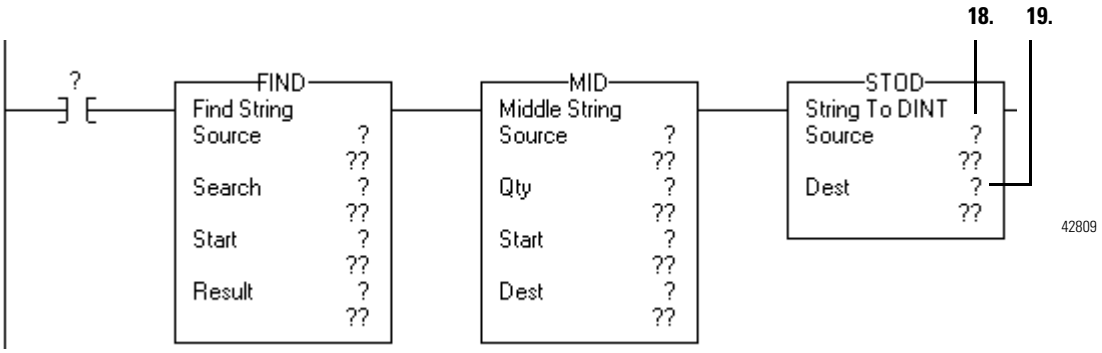
If:	Then:
STOR	Go to step 15.
STOD	Go to step 18.



15. Enter the tag that stores the value. (The tag from step 13.)

16. Enter a tag name to store the value for use in your application. Define the data type as REAL.

17. Go to step 20.



18. Enter the tag that stores the value. (The tag from step 13.)

19. Enter a tag name to store the value for use in your application. Define the data type as DINT.

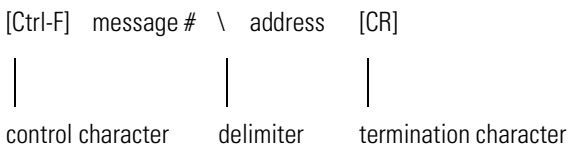
20. Does the string have another value that you want to use?

If:	Then:
yes	A. Add 1 to the Result of the Find instruction. (The tag from step 9.) B. Repeat steps 2 - 19.
no	Stop. You are done with this procedure.

Build a String

Use the following steps to build a string from variables in your application. You can then send the string to an ASCII triggered device, such as a MessageView terminal.

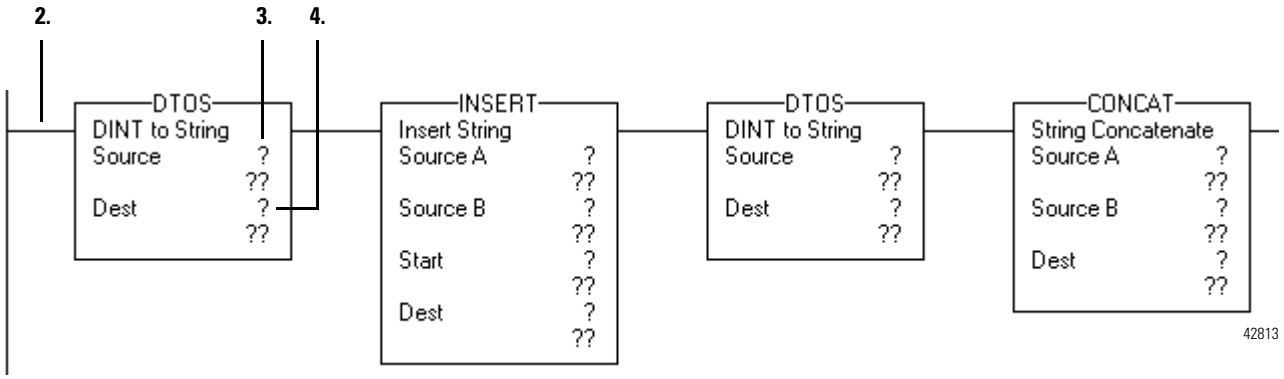
- In this procedure, you build a string that contains two variables. For example, an operator terminal may require a string that looks like this:



- If you need to include more variables, use additional INSERT or CONCAT instructions.
- If you need to send a floating-point value, use a RTOS instruction in place of the DTOS instruction.
- The final string will not include the termination character. When you send the string, use an AWA instruction to automatically append the termination character.

Steps:

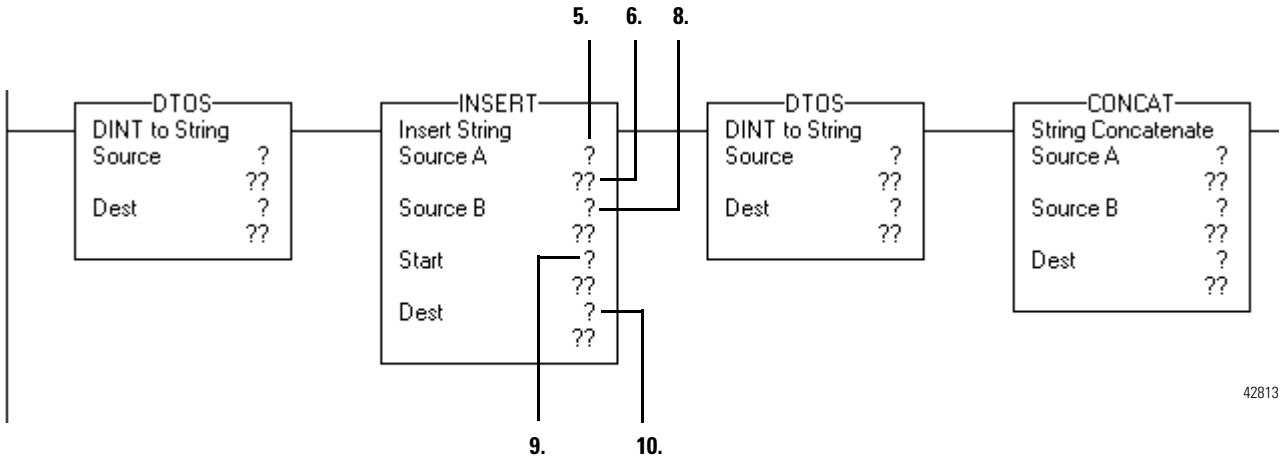
1. Enter the following rung:



2. Enter the input condition (s) that determines when to build the string.

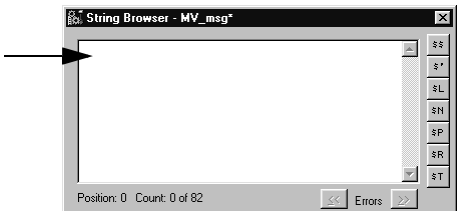
3. Enter the DINT tag that contains the first value for the string.

4. Enter a tag name to stores the ASCII representation of the value. Define the data type as a **string**.



5. Enter a tag name to store the control and delimiter characters for the string. Define the data type as a string.

6. Double-click the value area of the Source A.



7. Type the control character and delimiter and choose *OK*.

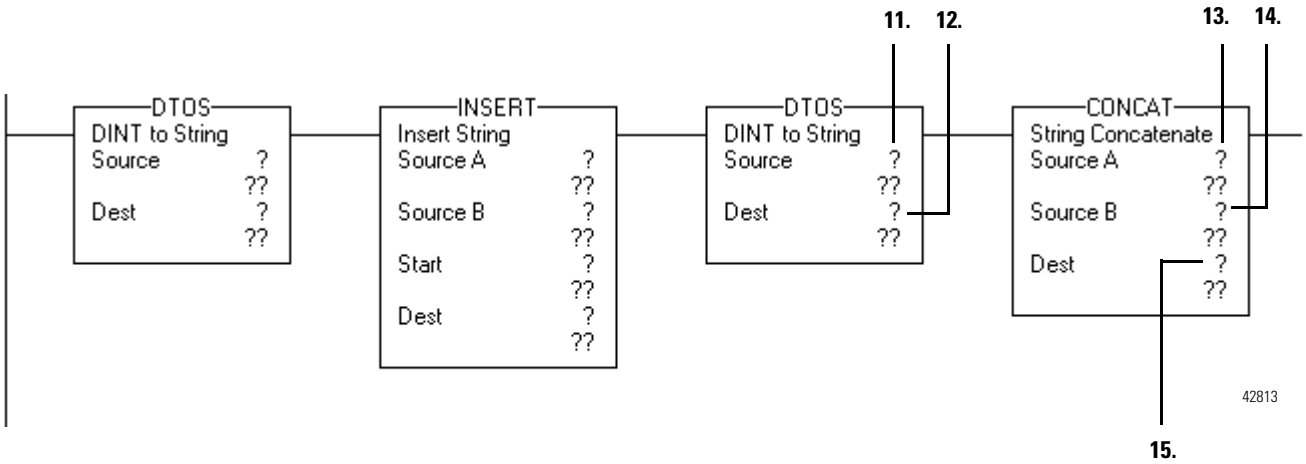
For a control character, type the hex code of the character. For a list of hex codes, see the back cover of this manual.

8. Enter the tag that stores the ASCII representation of the first value. (The tag from step 4.)

9. Enter 2.

This puts the value after the first character (control character) in Source A.

10. Enter a tag name to store the partially completed string. Define the data type as a string.



11. Enter the DINT tag that contains the second value for the string.

12. Enter a tag name to store the ASCII representation of the value. Define the data type as a string.

13. Enter the tag that stores the partially completed string. (The tag from step 10.)

14. Enter the tag that stores the ASCII representation of the second value. (The tag from step 12.)

15. Enter a tag name to store the completed string. Define the data type as a string.

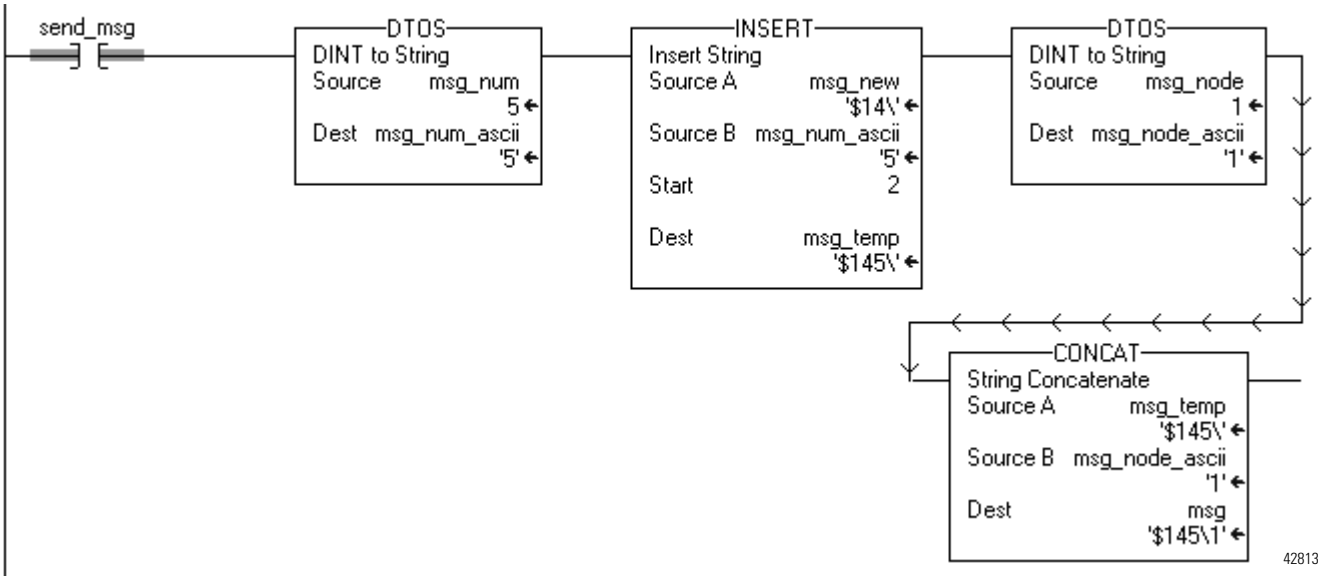
EXAMPLE

To trigger a message in a MessageView terminal, the controller sends the terminal a message in the following format: [Ctrl-T] message # \ address [CR]

When *send_msg* is on, the rung does the following:

- The first DTOS instruction converts the message number to ASCII characters.
- The INSERT instruction inserts the message number (in ASCII) after the control character [Ctrl-T]. (The hex code for Ctrl-T is $14.)
- The second DTOS instruction converts the node number of the terminal to ASCII characters.
- The CONCAT instruction puts the node number (in ASCII) after the backslash [\ ] and stores the final string in *msg*.

To send the message, an AWA instruction sends the *msg* tag and appends the carriage return [CR].



Notes:

Force Values

When to Force a Value

Use a force to override an input or output value:

If you want to override:	Then force the	Notes:
a value from the produced tag of another controller	consumed tag	Forcing an input or consumed tag: • overrides the value regardless of the value of the physical device or produced tag • does not affect the value received by other controllers monitoring that input or produced tag
a value from an input device	input data bit or value	
your logic and specify the value of a produced tag	produced tag	• Forcing an output or produced tag overrides the logic for the physical device or other controller (s). • Other controllers monitoring that output module in a listen-only capacity will also see the forced value.
your logic and specify the state an output device	output data bit or value	

When you force a value:

- You can force all I/O data, except for configuration data.
- If the tag is an array or structure, such as an I/O tag, force a BOOL, SINT, INT, DINT, or REAL element or member.
- If the data value is a SINT, INT, or DINT, you can force the entire value or you can force individual bits within the value. Individual bits can have a force status of:
  - no force
  - force on
  - force off
- You can also force an alias to an I/O structure member, produced tag, or consumed tag.
  - An alias tag shares the same data value as its base tag, so forcing an alias tag also forces the associated base tag.
  - Removing a force from an alias tag removes the force from the associated base tag.

ATTENTION

If forces are enabled and anything is forced, keep personnel away from the machine area. Forcing can cause unexpected machine motion that could injure personnel.

IMPORTANT Forcing increases logic execution time. The more values you force, the longer it takes to execute the logic.

IMPORTANT Forces are held by the controller and not by the programming workstation. Forces remain even if the programming workstation is disconnected.

Enter a Force

Use the Monitor Tags tab of the Tags window or use the Ladder window to enter forces.

Enter Forces from the Tags Window

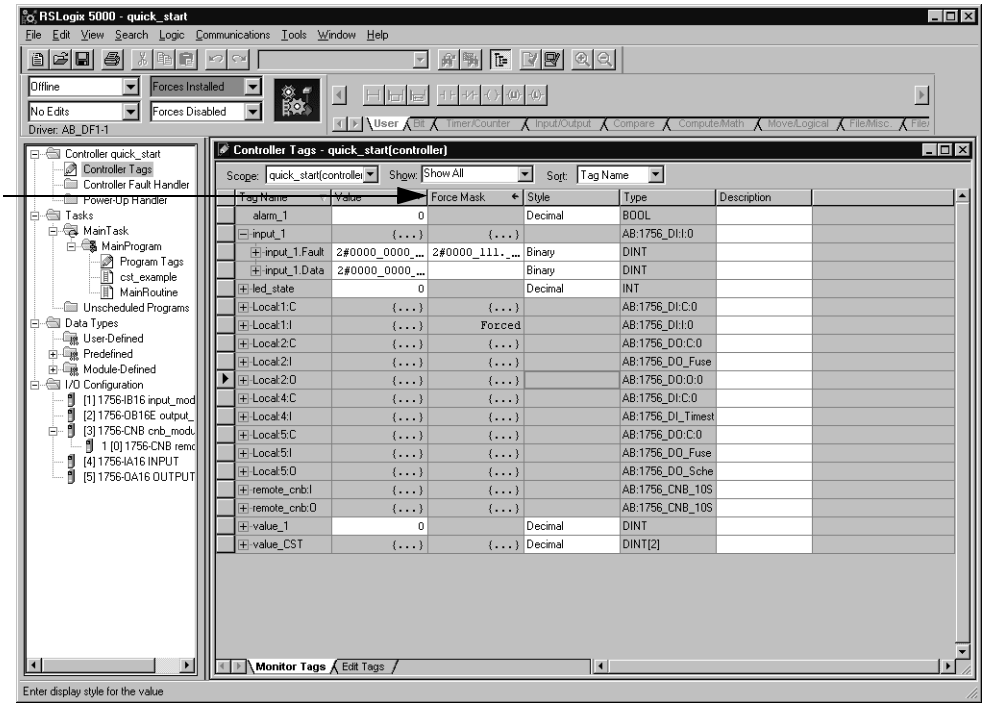
From the Tags window, Monitor Tags tab, you can force a value in two ways. You can:

- force a whole data value.

  For SINT, INT, DINT, and REAL values, you can force all the bits as one entity (the entire value)

- force the individual bits within a SINT, INT, or DINT value.

Enter force values in this column.

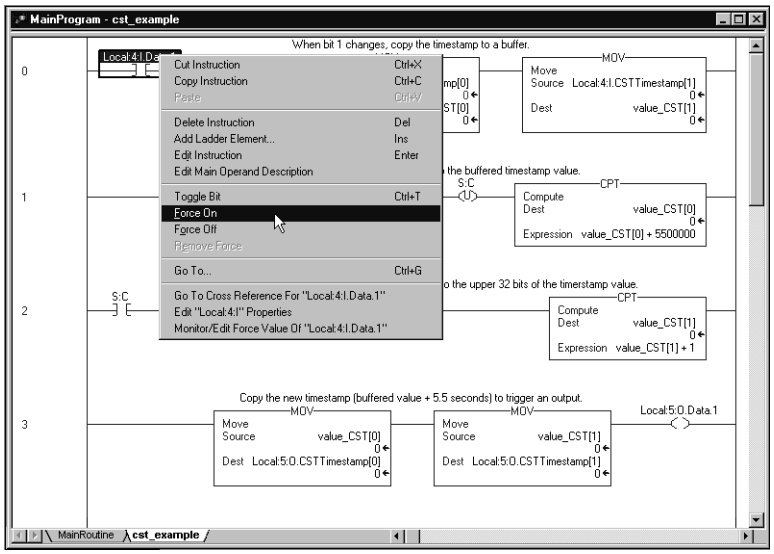


If you want to:	Do this:
force a whole SINT, INT, DINT, or REAL value	To force a whole value, type a force value in the Force Mask column, using a decimal, octal, hexadecimal, or float/exponential format. For a REAL value, you must use a float/exponential format. To remove a force for a whole value, type a space.
force bits within a value	To force an individual bit in a SINT, INT, or DINT value, expand the value and edit the Force Mask column. The force value is displayed in binary style, where: • "0" indicates force off • "1" indicates force on • "." indicates no force You can also use the bit pallet to select a bit to force.
force a BOOL	To force a BOOL, enter the force value, where: • "0" indicates force off • "1" indicates force on To remove a force, type a space.

Enter Forces from Ladder Logic

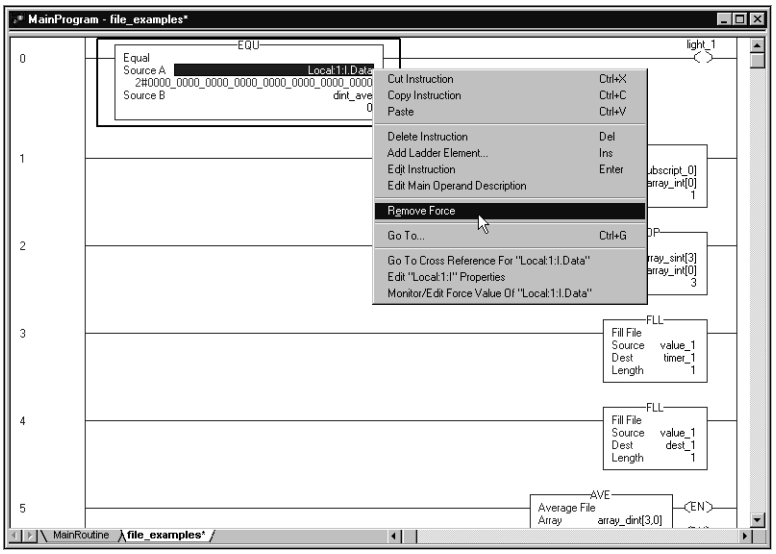
From ladder logic, you can set forces only for BOOL tags or integer bit values used in bit instructions.

Right-click on the BOOL tag or bit value.
Select **Force On**, **Force Off**, or **Remove Force**.



For forced values in the more complex instructions, you can only remove forces. You must use the data monitor to set force values for these values.

Right-click on the forced value.
Select **Remove Force**.



Enable Forces

For a force (s) to take affect, you enable forces. You can only enable and disable forces at the controller level. You cannot enable or disable forces for a specific module, tag collection, or tag element.

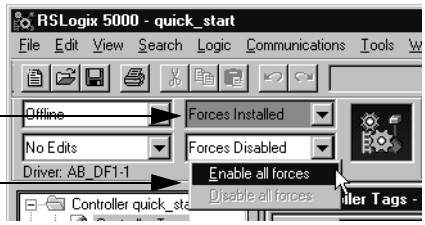
ATTENTION

Enabling forces causes input, output, produced, or consumed values to change. Keep personnel away from the machine area. Forcing can cause unexpected machine motion that could injure personnel.

You enable forces from the Online Bar.

Forces Installed indicates that force values have been entered.

Select **Enable all forces**.

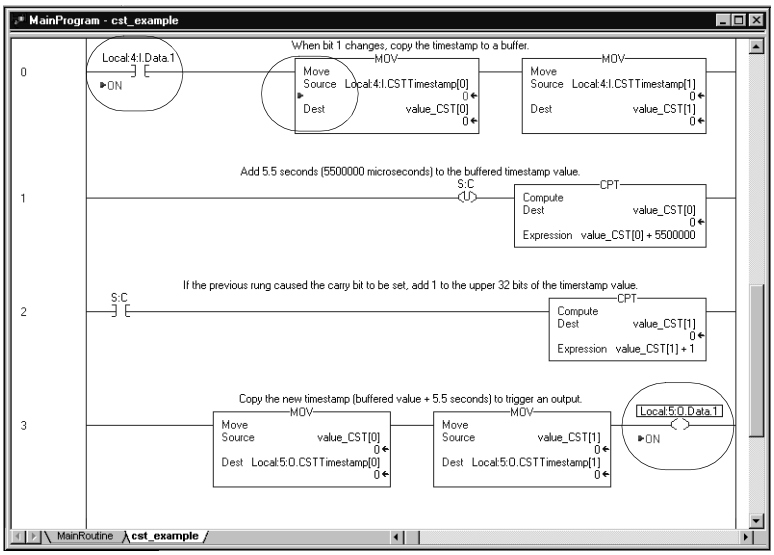


IMPORTANT

If you download a project that has forces enabled, the programming software prompts you to enable or disable forces after the download completes.

When forces are enabled, a > appears next to the forced value in the ladder editor.

When forces are enabled, the ladder editor indicates which forces are on.



Disable Forces

You can disable forces without removing forces from individual values or from the controller. By disabling forces, the project can execute as programmed. Forces are still entered, but they are not executed.

Select **Disable all forces**.

Remove Forces

You can remove forces from individual values or from the entire controller.

You can remove individual forces from the data monitor.

If you want to remove a force from a:	Do this:
whole SINT, INT, DINT, or REAL value	Right-click on the value in the data monitor and select **Remove Force**.
bits within a value	Expand the value and edit the Force Mask column. Change the bit value to "." to indicate no force.
BOOL value	Type a space.

If the force is on a BOOL tag or bit value, you can also remove forces from the ladder editor. Right-click on the value and select **Remove Force**.

If you remove each force individually, forces can still be enabled.

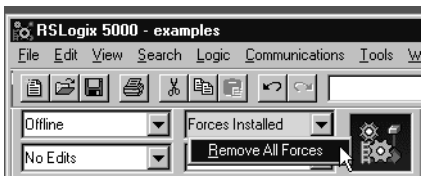
ATTENTION

If you have removed forces, but forces are still enabled and you set a force value, it takes affect immediately. Keep personnel away from the machine area. Forcing can cause unexpected machine motion that could injure personnel.

Removing a force on an alias tag also removes the force on the base tag.

At the controller level, you can remove all forces. Removing all forces disables forces and clears all force mask values.

Select **Remove all forces**.



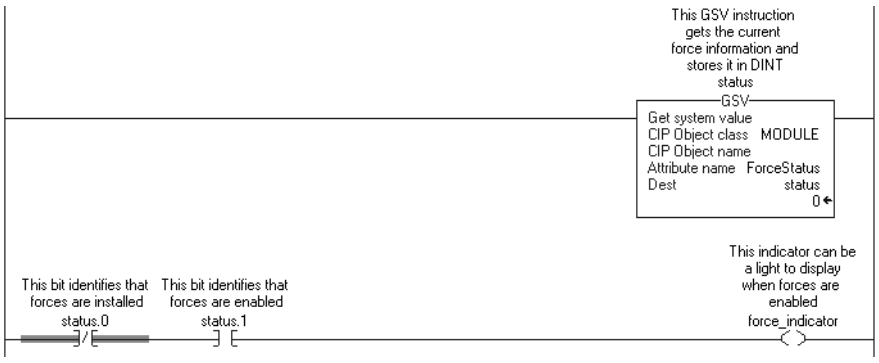
Monitor Forces

You can monitor force status in the following ways:

- RSLogix5000 software
- application logic
- FORCE LED. (Logix5550 controllers do not have an LED to indicate force status.)

If the FORCE LED is:	Then:
off	• No tags contain force values. • Forces are inactive (disabled).
flashing	• At least one tag contains a force value. • Force values are inactive (disabled).
solid	• Forces are active (enabled). • Force values may or may not exist.

The following example shows how to check whether forces are present and enabled and set your own LED indicator.


This GSV instruction gets the current force information and stores it in DINT status
GSV
Get system value
CIP Object class MODULE
CIP Object name
Attribute name ForceStatus
Dest status
0
This bit identifies that forces are installed
status.0
This bit identifies that forces are enabled
status.1
This indicator can be a light to display when forces are enabled
force_indicator

Notes:

Develop a Fault Routine

When to Use This Procedure

If a fault condition occurs that is severe enough for the controller to shut down, the controller generates a **major fault** and stops the execution of logic.

- Depending on your application, you may not want all major faults to shut down your entire system.
- In those situations, you can use a fault routine to clear a specific fault and let at least some of your system continue to operate.

EXAMPLE Use a fault routine

In a system that uses recipe numbers as indirect addresses, a miss-typed number could produce a major fault, such as type 4, code 20.

- To keep the entire system from shutting down, a fault routine clears any type 4, code 20, major faults.

Develop a Fault Routine

To clear a specific fault:

1. Create a user-defined data type that will store the fault information. Name the data type *FaultRecord* and assign the following members:

Name:	Data Type:	Style:
TimeLow	DINT	Decimal
TimeHigh	DINT	Decimal
Type	INT	Decimal
Code	INT	Decimal
Info	DINT[8]	Hex

2. Create a tag that will store the fault information. Select the *FaultRecord* data type.

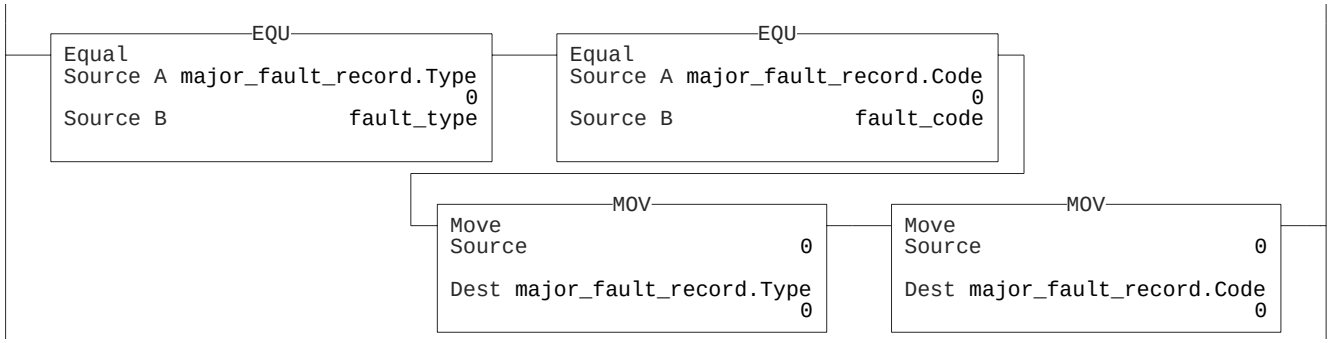
3. Which type of faults do you want the logic to clear?

For a fault due to:	Do this:
execution of an instruction	Create a fault routine for the program: A. In the controller organizer, right-click `name_of_program` and select *New Routine*. B. In the name box, type a name for the fault routine (`name_of_fault_routine`). C. From the *Type* drop-down list, select *Ladder*. D. Click *OK*. E. Right-click `name_of_program` and select *Properties*. F. Click the *Configuration* tab. G. From the *Fault* drop-down list, select `name_of_fault_routine.` H. Click *OK*. I. Double-click `name_of_fault_routine` .
power loss I/O task watchdog mode change motion axis	Create a program and main routine for the *Controller Fault Handler*. A. In the controller organizer, right-click *Controller Fault Handler* and select *New Program*. B. Type: • `name_of_program` • `description` (optional) C. Click *OK*. D. Click the + sign next to *Controller Fault Handler*. E. Right-click `name_of_program` and select *New Routine*. F. Type: • `name_of_routine` • `description` (optional) G. From the *Type* drop-down list, select the programming language for the routine. H. Click *OK*. I. Right-click `name_of_program` and select *Properties*. J. Click the *Configuration* tab. K. From the *Main* drop-down list, select `name_of_routine` L. Click *OK*. M. Double-click `name_of_routine.`

4. Enter the following logic:

Gets fault information and stores it in the *major_fault_record* tag (user-define structure)

If *fault_type, fault_code* occurs, sets the fault type and code in the *major_fault_record* tag to zero.



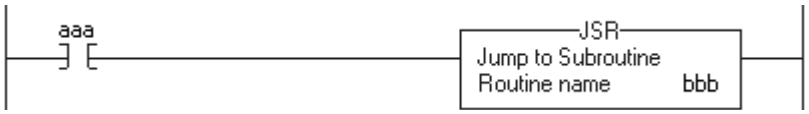
When the *MAJORFAULTRECORD* is set to the new values that are contained in the *major_fault_record* tag, the fault clears and the controller resumes execution.

where:	is the:
major_fault_record	tag from step 2.
fault_type	value for the fault type that you want to clear
fault_code	value for the fault code that you want to clear

Test a Fault Routine

You can use a JSR instruction to test the fault routine of a program without creating an error (i.e., simulate a fault):

1. Create a BOOL tag that you will use to initiate the fault.
2. In the main routine or a subroutine of the program, enter the following rung:

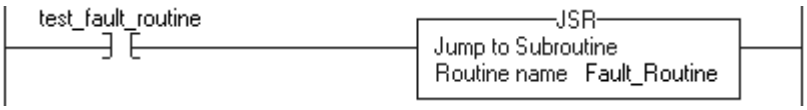


where:	is the:
aaa	tag that you will use to initiate the fault (Step 1.)
bbb	fault routine of the program

3. To simulate a fault, set the input condition.

EXAMPLE Test a fault routine

When *test_fault_routine* is on, a major fault occurs and the controller executes *Fault_Routine*.



Create a User-Defined Major Fault

When to Use this Procedure

If you want to suspend (shut down) the controller based on conditions in your application, create a user-defined major fault. With a user-defined major fault:

- You define a value for the fault code.
- The controller handles the fault the same as other major faults:
  - The controller changes to the **faulted mode** (major fault) and stops executing the logic.
  - Outputs are set to their configured state or value for faulted mode.

EXAMPLE User-defined major fault

When *input_value* is greater than 80, produce a major fault and generate a fault code of 999.

Create a User-Defined Major Fault

1. Does a fault routine already exist for the program?

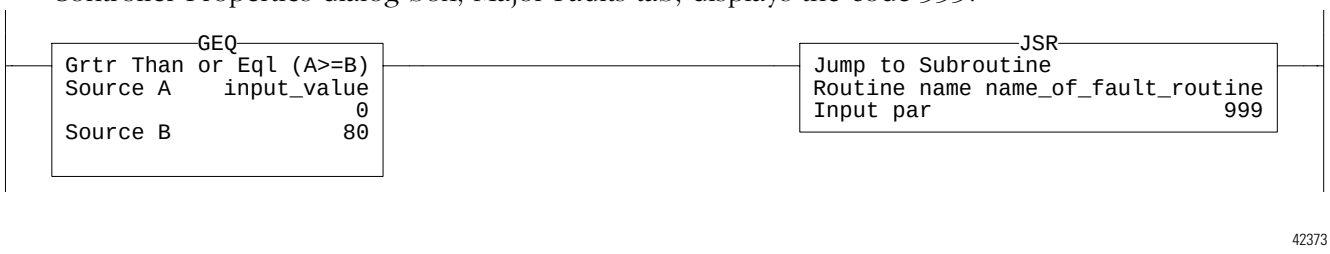
If:	Then:
Yes	Go to step 2.
No	Create a fault routine for the program: A. In the controller organizer, right-click `name_of_program` and select *New Routine*. B. In the name box, type a name for the fault routine (`name_of_fault_routine`). C. From the *Type* drop-down list, select *Ladder*. D. Click *OK*. E. Right-click `name_of_program` and select *Properties*. F. Click the *Configuration* tab. G. From the *Fault* drop-down list, select `name_of_fault_routine.` H. Click *OK*. I. Double-click `name_of_fault_routine.` J. Enter an NOP instruction (so the routine verifies without an error).

2. In the main routine of the program, enter the following rung:

where:	is:
name_of_fault_routine	routine from step 1.
x	value for the fault code

EXAMPLE Create a user-defined major fault

When *input_value* is greater than or equal to 80, execution jumps to *name_of_fault_routine*. A major fault occurs and the controller enters the faulted mode. Outputs go to the faulted state. The Controller Properties dialog box, Major Faults tab, displays the code *999*.



Monitor Minor Faults

When to Use This Procedure

If a fault condition occurs that is *not* severe enough for the controller to shut down, the controller generates a **minor fault**.

- The controller continues to execute.
- You do not need to clear a minor fault.
- To optimize execution time and ensure program accuracy, you should monitor and correct minor faults.

Monitor Minor Faults

To use ladder logic to capture information about a minor fault:

To check for a:	Do this:
periodic task overlap	1. Enter a GSV instructions that gets the *FAULTLOG* object, *MinorFaultBits* attribute. 2. Monitor bit 6.
load from nonvolatile memory	1. Enter a GSV instructions that gets the *FAULTLOG* object, *MinorFaultBits* attribute. 2. Monitor bit 7.
problem with the serial port	1. Enter a GSV instructions that gets the *FAULTLOG* object, *MinorFaultBits* attribute. 2. Monitor bit 9.
low battery	1. Enter a GSV instructions that gets the *FAULTLOG* object, *MinorFaultBits* attribute. 2. Monitor bit 10.
problem with an instruction	1. Create a user-defined data type that stores the fault information. Name the data type *FaultRecord* and assign the following members:

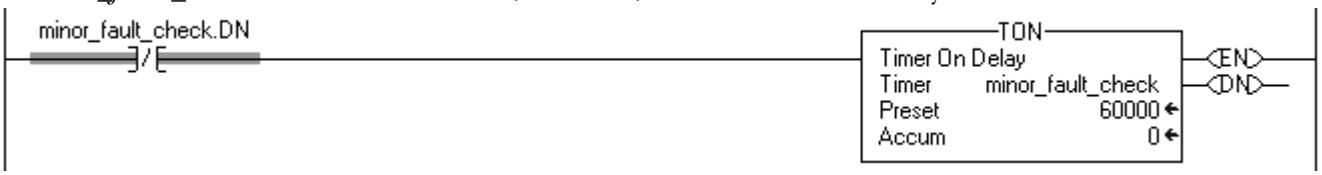
Name:	Data Type:	Style:
TimeLow	DINT	Decimal
TimeHigh	DINT	Decimal
Type	INT	Decimal
Code	INT	Decimal
Info	DINT[8]	Hex

2. Create a tag that will store the values of the *MinorFaultRecord* attribute. Select the data type from step 1.
3. Monitor *S:MINOR*.
4. If *S:MINOR* is on, use a GSV instruction to get the values of the *MinorFaultRecord* attribute.
5. If you want to detect a minor fault that is caused by another instruction, reset *S:MINOR*. (S:MINOR remains set until the end of the scan.)

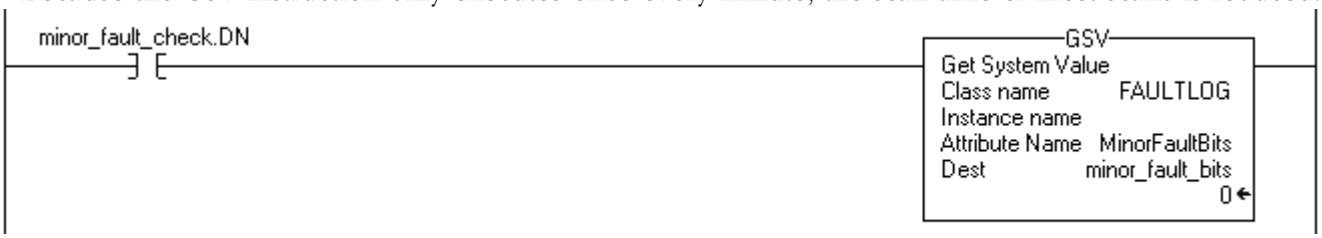
The following example checks for a low battery warning.

EXAMPLE Check for a minor fault

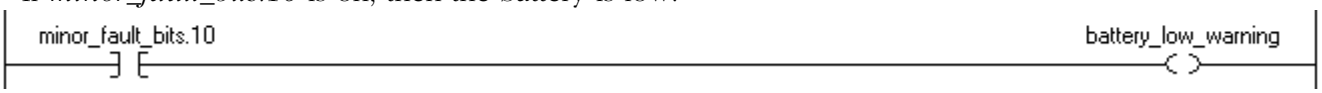
Minor_fault_check times for 1 minute (60000 ms) and then automatically restarts itself.



Every minute, *minor_fault_check.DN* turns on for one scan. When this occurs, the GSV instruction gets the value of the *FAULTLOG* object, *MinorFaultBits* attribute, and stores it in the *minor_fault_bits* tag. Because the GSV instruction only executes once every minute, the scan time of most scans is reduced.



If *minor_fault_bits.10* is on, then the battery is low.



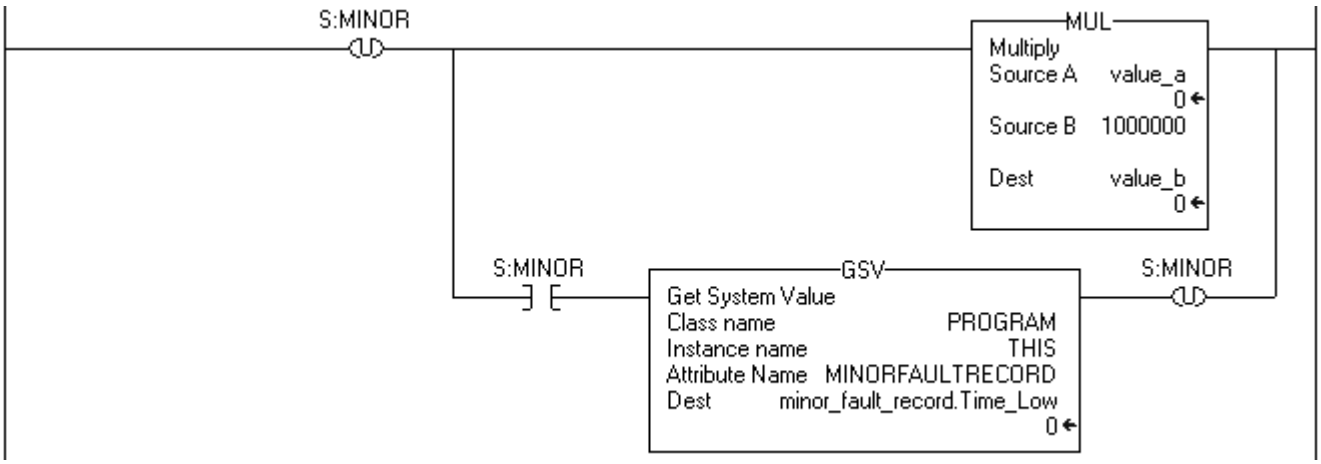
42373

The following example checks for a minor fault that is caused by a specific instruction.

EXAMPLE Check for a minor fault that is caused by an instruction

Multiplies *value_a* by 1000000 and checks for a minor fault, such as a math overflow:

- To make sure that a previous instruction did not produce the fault, the rung first clears S:MINOR.
- The rung then executes the multiply instruction.
- If the instruction produces a minor fault, the controller sets S:MINOR.
- If S:MINOR is set, the GSV instruction gets information about the fault and resets S:MINOR.



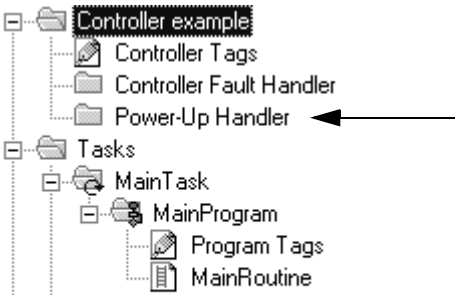
42373

Notes:

Develop a Power-Up Routine

When to Use This Procedure

The **power-up handler** is an optional task that executes when the controller powers up in the Run mode.



Use the power-up handler when you want to accomplish either of the following after power is lost and then restored:

- Prevent the controller from returning to Run mode.
  - The power-up handler will produce a major fault, type 1, code 1, and the controller will enter the Faulted mode.
- Take specific actions and then resume normal execution of the logic.

Develop a Power-Up Routine

The steps to develop a power-up routine are similar to the steps to develop a fault routine:

1. Create a user-defined data type that will store the fault information. Name the data type *FaultRecord* and assign the following members:

Name:	Data Type:	Style:
TimeLow	DINT	Decimal
TimeHigh	DINT	Decimal
Type	INT	Decimal
Code	INT	Decimal
Info	DINT[8]	Hex

2. Create a tag that will store the fault information. Select the *FaultRecord* data type.

3. Create a program for the Power-Up Handler:

Action:	Detailed steps:
1. Create a program.	A. In the controller organizer, right-click *Power-Up Handler* and select *New Program*. B. Type: • `name_of_program` • `description` (optional) C. Click *OK*.
2. Create and assign a **main routine** (the routine to execute first in the program).	A. Click the + sign that is next to *Power-Up Handler*. B. Right-click `name_of_program` and select *New Routine*. C. Type: • `name_of_main_routine` • `description` (optional) D. From the *Type* drop-down list, select the programming language for the routine. E. Click *OK*. F. Right-click `name_of_program` and select *Properties*. G. Click the *Configuration* tab. H. From the *Main* drop-down list, select `name_of_main_routine` I. Click *OK*. J. To add additional routines (subroutines) to the program, repeat steps B. to E.

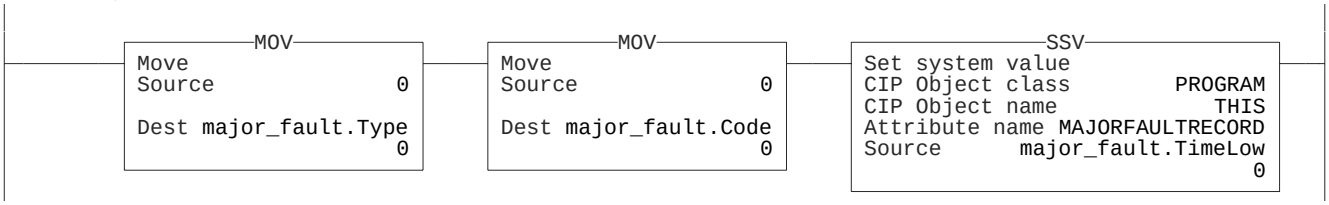
3. How do you want to handle a power loss?

To:	Do this:
Prevent the controller from returning to Run mode	You are done. When power is restored, a major fault, type 1, code 1, will occur and the controller will enter the Faulted mode.
When power is restored, take specific actions and then resume normal operation	A. Open (double-click) `name_of_routine.` B. Enter the logic for the actions.

4. Enter the following logic to clear the fault:

Gets fault information and stores it in the *major_fault* tag (user-define structure)

Sets the fault type and code in the *major_fault* tag to zero and sets *MAJORFAULTRECORD* to the new values, which clears the fault.



where:

major_fault is the tag from step 2.

Notes:

Store and Load a Project Using Nonvolatile Memory

When to Use This Procedure

Currently, only 1756-L55M23 and -L55M24 controllers have nonvolatile memory for project storage. In the future, we will add the feature to additional controllers within the Logix5000 family of controllers.

IMPORTANT Nonvolatile memory stores the contents of the user memory at the time that you store the project.

- Changes that you make after you store the project are *not* reflected in nonvolatile memory.
- If you want to store changes such as online edits, tag values, or a ControlNet network schedule, store the project again after you make the changes.

Use this procedure to **store** or **load** a project using the **nonvolatile memory** of a controller.

- If the controller loses power and does not have enough battery capacity, it loses the project in user memory.
- Nonvolatile memory lets you keep a copy of your project on the controller. The controller does not need power to keep this copy.
- You can load the copy from nonvolatile memory to the user memory of the controller:
  - on every power-up
  - whenever there is no project in the controller and it powers-up
  - anytime through RSLogix 5000 software

A store or load has the following parameters:

Table 19.A Parameters of a store or load

Parameter:	Store:	Load:
How much time does a store or load take?	several minutes	several seconds
In what controller mode (s) can I store or load a project?	program mode	
Can I go online with the controller during a store or load?	no	
What is the state of the I/O during a store or load?	I/O remains in its configured state for program mode.	

How to Use This Procedure

If you want to:	Then:
store a project in the nonvolatile memory of the controller	Go to "Store a Project" on page 19-3.
overwrite the current project in the controller with the project that is stored in the nonvolatile memory of the controller	Go to "Load a Project" on page 19-6.
load the project after a power loss cleared the memory because there was no battery	
use ladder logic to flag that your project loaded from nonvolatile memory	Go to "Check for a Load" on page 19-9.
remove a project from the nonvolatile memory of the controller	Go to "Clear Nonvolatile Memory" on page 19-10.

Store a Project

In this task, you store a project in the nonvolatile memory of the controller. This overwrites a project that is currently in the nonvolatile memory.

ATTENTION

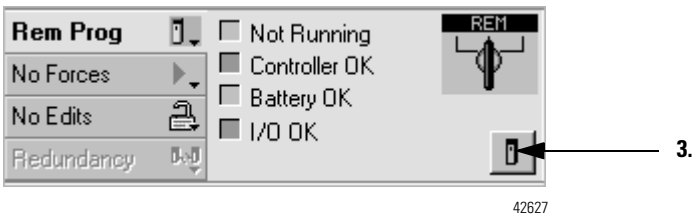
During a store, all active servo axes are turned off. Before you store a project, make sure that this *will not* cause any unexpected movement of an axis.

Before you store the project:

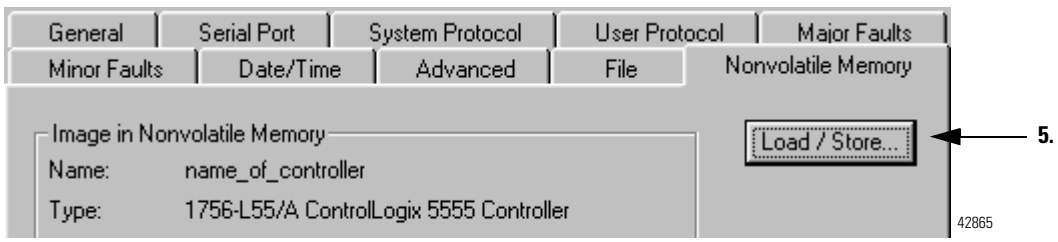
- make all the required edits to the logic
- download the project to the controller

Steps:

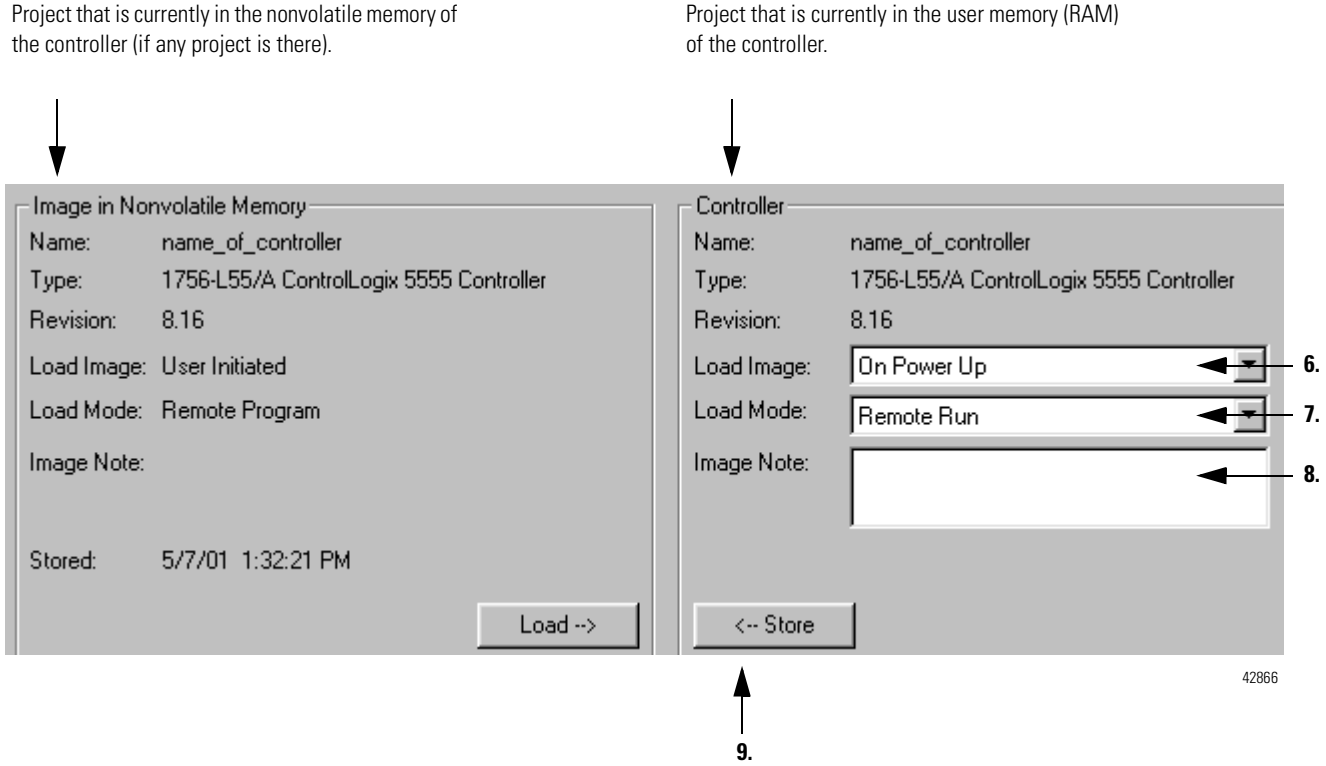
1. Go online with the controller.
2. Put the controller in Program mode (Rem Program or Program).



3. On the Online toolbar, click the controller properties button.
4. Click the *Nonvolatile Memory* tab.



5. Choose *Load/Store*.



6. When (under what conditions) do you want to load the project back into the user memory (RAM) of the controller?

If you want to load it:	Then select:	Notes:
whenever you turn on or cycle the chassis power	*On Power Up*	• During a power cycle, you will lose any online changes, tag values, and network schedule that you have not stored in the nonvolatile memory. • With this option, the project also loads when you update the firmware of the controller. After the load, the controller automatically goes to the mode that you select from the Load Mode drop-down list (step 7). • You can always use RSLogix 5000 software to load the project.
whenever there is no project in the controller and you turn on or cycle the chassis power	*On Corrupt Memory*	• With this option, the project also loads when you update the firmware of the controller. After the load, the controller automatically goes to the mode that you select from the Load Mode drop-down list (step 7). • You can always use RSLogix 5000 software to load the project.
only through RSLogix 5000 software	*User Initiated*	

7. In step 6, which load image option did you select?

If:	Then:
On Power Up	Select the mode that you want the controller to go to after a load: • remote program • remote run To go to this mode after a load, turn the keyswitch of the controller to the REM position.
On Corrupt Memory	
User Initiated	Go to step 8.

8. Type a note that describes the project that you are storing, if desired.

9. Choose <– *Store*.

A dialog box asks you to confirm the store.

10. To store the project, choose *Yes*.

During the store, the following events occur:

- On the front of the controller, the OK LED displays the following sequence:
  flashing green ⇒ red ⇒ green
- RSLogix 5000 software goes offline.
- A dialog box tells you that the store is in progress.

11. Choose *OK*.

When the store is finished, you remain offline. If you want to be online, you must manually go online.

Load a Project

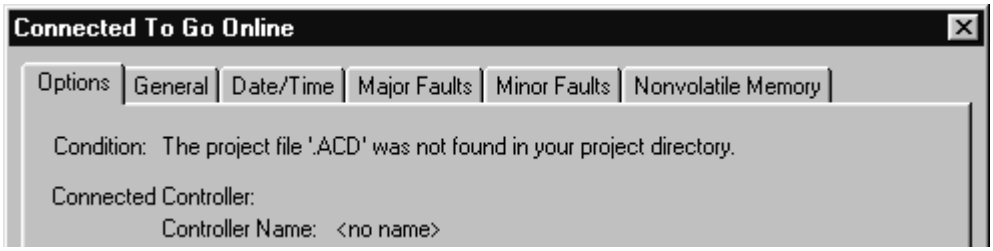
In this task, you use RSLogix 5000 software to load the project from nonvolatile memory.

ATTENTION

During a load, all active servo axes are turned off. Before you load a project, make sure that this *will not* cause any unexpected movement of an axis.

Steps:

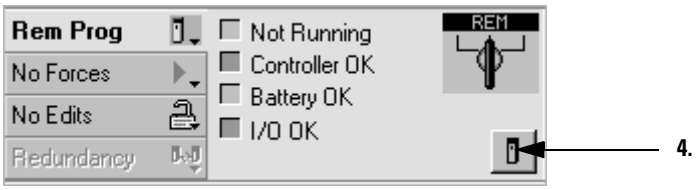
1. Go online with the controller.
2. Did the following dialog box open?



42873

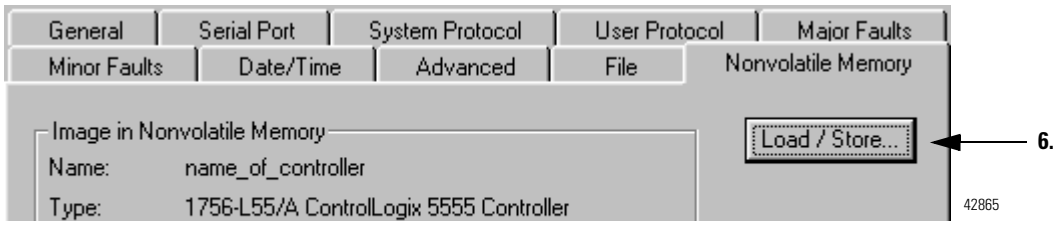
If:	Then:
No	Go to step 3.
Yes	Go to step 5.

3. Put the controller in Program mode (Rem Program or Program).

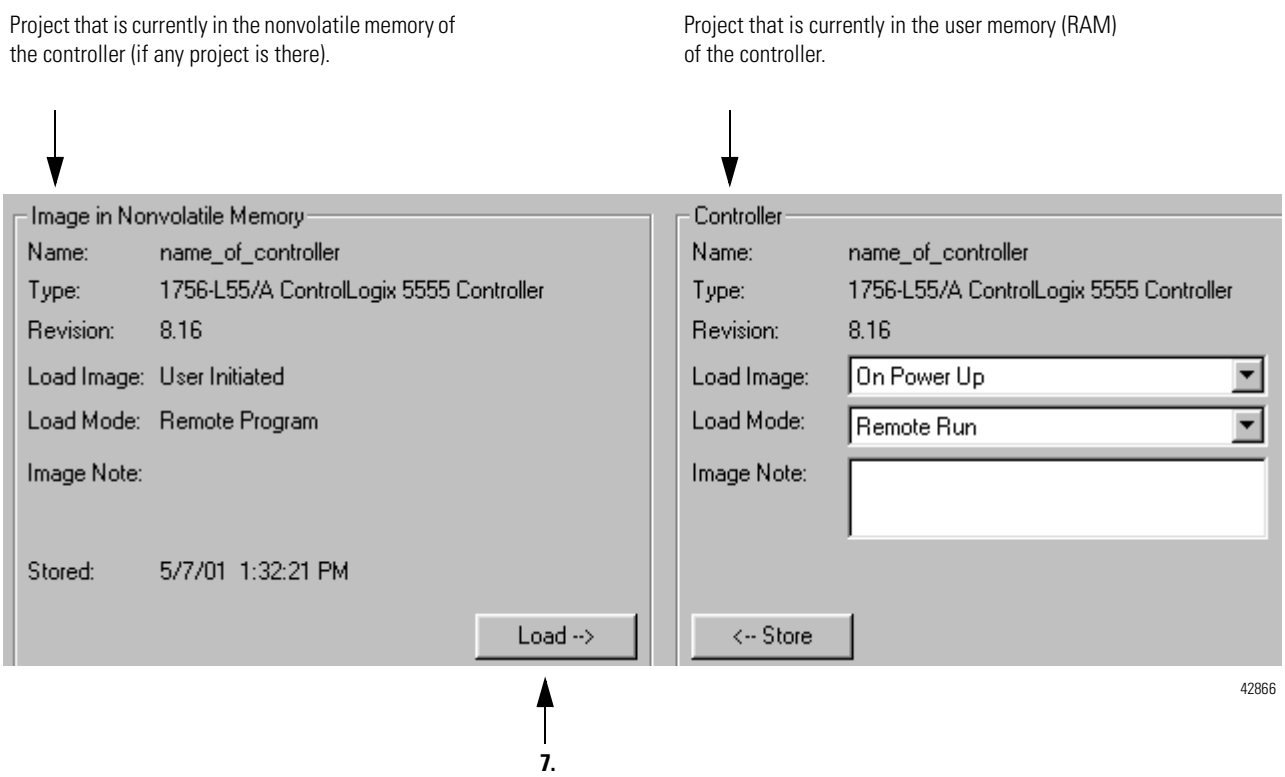


42627

4. On the Online toolbar, click the controller properties button.
5. Click the *Nonvolatile Memory* tab.



6. Choose *Load/Store*.



7. Choose *Load –>*.

A dialog box asks you to confirm the load.

8. To load the project from the nonvolatile memory, choose *Yes*.

During the load, the following events occur:

- On the front of the controller, the OK LED displays the following sequence:

  red ⇒ green

- RSLogix 5000 software goes offline.

When the load is finished, you remain offline. If you want to be online, you must manually go online.

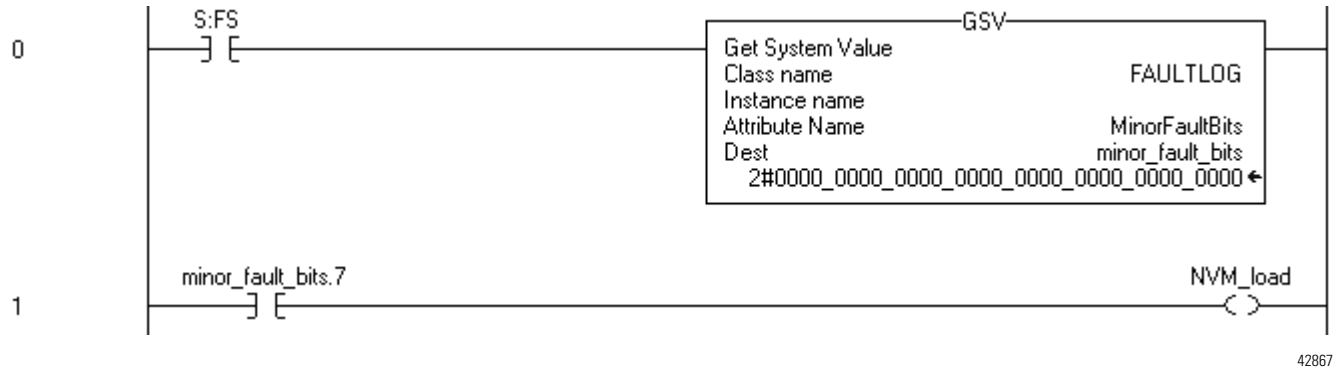
Check for a Load

When the controller loads a project from nonvolatile memory, it provides the following information:

- logs a minor fault (type 7, code 49)
- sets the FAULTLOG object, MinorFaultBits attribute, bit 7

If you want your project to flag that it loaded from nonvolatile memory, use the following ladder logic:

On the first scan of the project (*S:FS* is on), the GSV instruction gets the FAULTLOG object, MinorFaultBits attribute, and stores the value in *minor_fault_bits*. If bit 7 is on, the controller loaded the project from its nonvolatile memory.



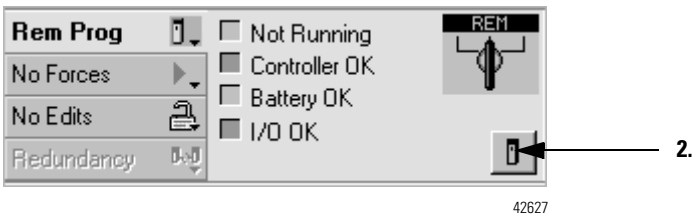
Where:	Is:
minor_fault_bits	Tag that stores the FAULTLOG object, MinorFaultBits attribute. Data type is DINT.
NVM_load	Tag that indicates that the controller loaded the project from its nonvolatile memory.

Clear Nonvolatile Memory

Typically, you do not have to clear the nonvolatile memory of the controller. When you store a project, you overwrite the complete contents of the nonvolatile memory.

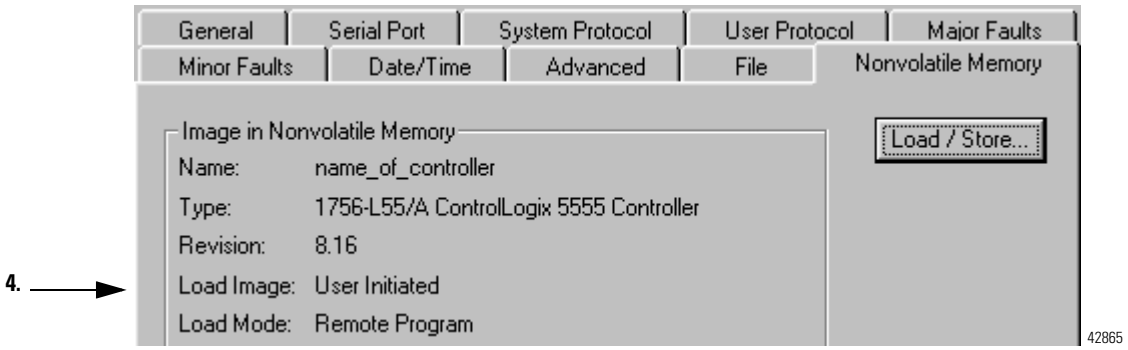
If you want to remove a project from the nonvolatile memory, complete the following steps:

1. Go online with the controller.



2. On the Online toolbar, click the controller properties button.

3. Click the *Nonvolatile Memory* tab.



4. Is the *Load Image* option set to *User Initiated*?

If:	Then:
No	Go to step 5.
Yes	Go to step 11.

5. Choose *Load/Store*.

6. In the *Load Image* drop-down list, select *User Initiated*.

7. Choose <- *Store*.

A dialog box asks you to confirm the store.

8. To store the project, choose *Yes*.

A dialog box tells you that the store is in progress.

9. Choose *OK*.

10. Wait until the OK LED on the front of the controller is steady green. This indicates that the store is finished.

11. Disconnect the battery from the controller.

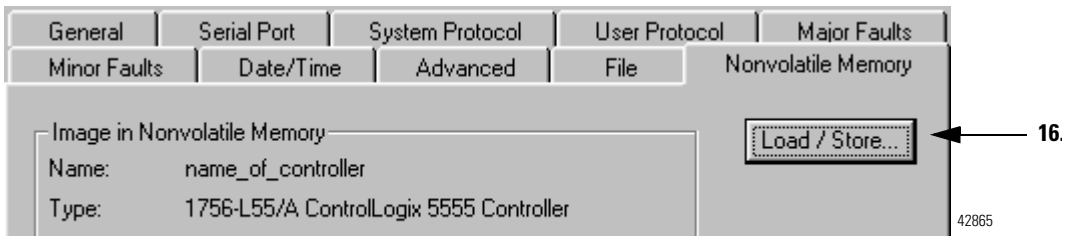
12. Cycle the power to the chassis.

13. Re-connect the battery to the controller.

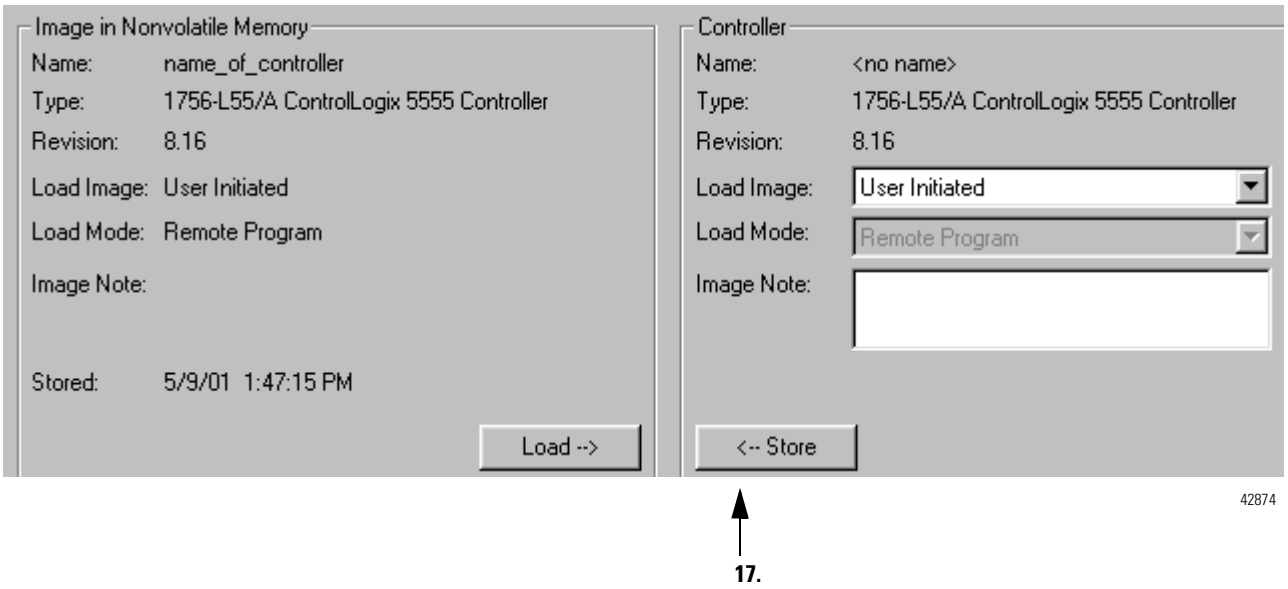
14. Go online with the controller.

The Connected To Go Online dialog box opens.

15. Click the *Nonvolatile Memory* tab.



16. Choose *Load/Store*.



17. Choose <– *Store*.

A dialog box asks you to confirm the store.

18. To store the project, choose *Yes*.

During the store, the following events occur:

- On the front of the controller, the OK LED displays the following sequence:
  flashing green ⇒ red ⇒ green
- RSLogix 5000 software goes offline.
- A dialog box tells you that the store is in progress.

19. Choose *OK*.

When the store is finished, you remain offline. If you want to be online, you must manually go online.

Fault Codes

When to Use This Appendix

Use this appendix to interpret:

- Major Fault Codes
- Minor Fault Codes

Major Fault Codes

Use the following table to determine the cause and corrective action for a major fault. The type and code correspond to the type and code displayed in these locations:

- Controller Properties dialog box, Major Faults tab
- PROGRAM object, MAJORFAULTRECORD attribute

Table A.1 Major Fault Types and Codes

Type:	Code:	Cause:	Recovery Method:
1	1	The controller powered on in Run mode.	Execute the power-loss handler.
3	16	A required I/O module connection failed.	Check that the I/O module is in the chassis. Check electronic keying requirements. View the controller properties Major Fault tab and the module properties Connection tab for more information about the fault.
3	20	Possible problem with the ControlBus chassis.	Not recoverable - replace the chassis.
3	23	At least one required connection was not established before going to Run mode.	Wait for the controller I/O light to turn green before changing to Run mode.
4	16	Unknown instruction encountered.	Remove the unknown instruction. This probably happened due to a program conversion process.
4	20	Array subscript too big, control structure .POS or .LEN is invalid.	Adjust the value to be within the valid range. Don't exceed the array size or go beyond dimensions defined.
4	21	Control structure .LEN or .POS < 0.	Adjust the value so it is > 0.
4	31	The parameters of the JSR instruction do not match those of the associated SBR or RET instruction.	Pass the appropriate number of parameters. If too many parameters are passed, the extra ones are ignored without any error.
4	34	A timer instruction has a negative preset or accumulated value.	Fix the program to not load a negative value into timer preset or accumulated value.
4	42	JMP to a label that did not exist or was deleted.	Correct the JMP target or add the missing label.
4	83	The data tested was not inside the required limits.	Modify value to be within limits.
4	84	Stack overflow.	Reduce the subroutine nesting levels or the number of parameters passed.

Table A.1 Major Fault Types and Codes (Continued)

Type:	Code:	Cause:	Recovery Method:
6	1	Task watchdog expired. User task has not completed in specified period of time. A program error caused an infinite loop, or the program is too complex to execute as quickly as specified, or a higher priority task is keeping this task from finishing.	Increase the task watchdog, shorten the execution time, make the priority of this task "higher," simplify higher priority tasks, or move some code to another controller.
7	40	Store to nonvolatile memory failed.	1. Try again to store the project to nonvolatile memory. 2. If the project fails to store to nonvolatile memory, replace the memory board.
7	42	Load from nonvolatile memory failed because the firmware revision of the project in nonvolatile memory does not match the firmware revision of the controller.	Update the controller firmware to the same revision level as the project that is in nonvolatile memory.
8	1	Attempted to place controller in Run mode with keyswitch during download.	Wait for the download to complete and clear fault.
11	1	Actual position has exceeded positive overtravel limit.	Move axis in negative direction until position is within overtravel limit and then execute Motion Axis Fault Reset.
11	2	Actual position has exceeded negative overtravel limit.	Move axis in positive direction until position is within overtravel limit and then execute Motion Axis Fault Reset.
11	3	Actual position has exceeded position error tolerance.	Move the position within tolerance and then execute Motion Axis Fault Reset.
11	4	Encoder channel A, B, or Z connection is broken.	Reconnect the encoder channel then execute Motion Axis Fault Reset.
11	5	Encoder noise event detected or the encoder signals are not in quadrature.	Fix encoder cabling then execute Motion Axis Fault Reset.
11	6	Drive Fault input was activated.	Clear Drive Fault then execute Motion Axis Fault Reset.
11	7	Synchronous connection incurred a failure.	First execute Motion Axis Fault Reset. If that doesn't work, pull servo module out and plug back in. If all else fails replace servo module.
11	8	Servo module has detected a serious hardware fault.	Replace the module.
11	9	Asynchronous Connection has incurred a failure.	First execute Motion Axis Fault Reset. If that doesn't work, pull servo module out and plug back in. If all else fails replace servo module.
11	32	The motion task has experienced an overlap.	The group's course update rate is too high to maintain correct operation. Clear the group fault tag, raise the group's update rate, and then clear the major fault.

Minor Fault Codes

Use the following table to determine the cause and corrective action for a minor fault. The type and code correspond to the type and code displayed in these locations:

- Controller Properties dialog box, Minor Faults tab
- PROGRAM object, MINORFAULTRECORD attribute

Table A.2 Minor Fault Types and Codes

Type:	Code:	Cause:	Recovery Method:
4	4	An arithmetic overflow occurred in an instruction.	Fix program by examining arithmetic operations (order) or adjusting values.
4	7	The GSV/SSV destination tag was too small to hold all of the data.	Fix the destination so it has enough space.
4	35	PID delta time ≤ 0.	Adjust the PID delta time so that it is > 0.
4	36	PID setpoint out of range	Adjust the setpoint so that it is within range.
4	51	The LEN value of the string tag is greater than the DATA size of the string tag.	1. Check that no instruction is writing to the LEN member of the string tag. 2. In the LEN value, enter the number of characters that the string contains.
4	52	The output string is larger than the destination.	Create a new string data type that is large enough for the output string. Use the new string data type as the data type for the destination.
4	53	The output number is beyond the limits of the destination data type.	Either: • Reduce the size of the ASCII value. • Use a larger data type for the destination.
4	56	The Start or Quantity value is invalid.	1. Check that the Start value is between 1 and the DATA size of the Source. 2. Check that the Start value plus the Quantity value is less than or equal to the DATA size of the Source.
4	57	The AHL instruction failed to execute because the serial port is set to no handshaking.	Either: • Change the Control Line setting of the serial port. • Delete the AHL instruction.
6	2	Periodic task overlap. Periodic task has not completed before it is time to execute again.	Simplify program(s), or lengthen period, or raise relative priority, etc.
7	49	Project loaded from nonvolatile memory.	
9	0	Unknown error while servicing the serial port.	Contact GTS personnel.
9	1	The CTS line is not correct for the current configuration.	Disconnect and reconnect the serial port cable to the controller. Make sure the cable is wired correctly

Table A.2 Minor Fault Types and Codes (Continued)

Type:	Code:	Cause:	Recovery Method:
9	2	Poll list error. A problem was detected with the DF1 master's poll list, such as specifying more stations than the size of the file, specifying more then 255 stations, trying to index past the end of the list, or polling the broadcast address (STN #255).	Check for the following errors in the poll list: • total number of stations is greater than the space in the poll list tag • total number of stations is greater than 255 • current station pointer is greater than the end of the poll list tag • a station number greater than 254 was encountered
9	5	DF1 slave poll timeout. The poll watchdog has timed out for slave. The master has not polled this controller in the specified amount of time.	Determine and correct delay for polling.
9	9	Modem contact was lost. DCD and/or DSR control lines are not being received in proper sequence and/or state.	Correct modem connection to the controller.
10	10	Battery not detected or needs to be replaced.	Install new battery.

IEC61131-3 Compliance

Using This Appendix

For information about:	See page:
Operating System	B-2
Data Definitions	B-2
Programming Languages	B-3
Instruction Set	B-4
IEC61131-3 Program Portability	B-4
IEC Compliance Tables	B-5

Introduction

The International Electrotechnical Commission (IEC) has developed a series of specifications for programmable controllers. These specifications are intended to promote international unification of equipment and programming languages for use in the controls industry. These standards provide the foundation for Logix5000 controllers and RSLogix 5000 programming software.

The IEC programmable controller specification is broken down into five separate parts each focusing on a different aspect of the control system:

- Part 1: General Information
- Part 2: Equipment and Requirements Test
- Part 3: Programming Languages
- Part 4: User Guidelines
- Part 5: Messaging Service Specification

The controls industry as a whole has focused on part 3 (IEC61131-3), Programming Languages, because it provides the cornerstone for implementing the other standards and provides the most significant end user benefit by reducing training cost. Because of this, only IEC61131-3 is addressed here.

The IEC61131-3 programming language specification addresses numerous aspects of programmable controller including the operating system execution, data definitions, programming languages, and instruction set. Components of the IEC61131-3 specification are categorized as required by the specification, optional or extensions. By so doing, the IEC61131-3 specification provides a minimum set of functionality that can be extended to meet end user application needs. The downside of this approach is that each programmable control system vendor may implement different components of the specification or provide different extensions.

Operating System

The preemptive, multitasking operating system (OS) of Logix5000 controllers complies with the IEC61131-3 definition. In IEC61131-3, the programmable controllers OS can contain zero or more tasks, that can execute one or more programs each containing one or more functions or routines. According to IEC61131-3, the number of each of these components is implementation dependent. Logix5000 controllers provide multiple tasks, each containing multiple programs and an unlimited number of functions or routines.

IEC61131-3 provides an option for creating different task execution classifications. Tasks may be configured as continuous, periodic, or event based. A continuous task does not need to be scheduled in that it will utilize any left over processing time when other tasks are dormant. Periodic tasks are scheduled to operate based on a reoccurring time period. The IEC61131-3 specification does not specify a time base for periodic task configuration. An IEC61131-3 event based task is triggered upon detection of the rising edge of a configured input. Logix5000 controllers support both continuous and periodic tasks. Additionally, the period for a periodic task is configurable starting as low as 1 millisecond (ms).

Data Definitions

The IEC61131-3 specification provides access to memory through the creation of named variables. IEC61131-3 names for variables consist of a minimum of six characters (RSLogix5000 programming software supports a minimum of 1 character) starting with an underscore "_" or an alpha character (A-Z), followed by one or more characters consisting of an underscore "_", alpha character (A-Z) or a number (0-9). Optionally, lower case alpha characters (a-z) can be supported as long as they are case insensitive (A = a, B = b, C = c …). Logix5000 controllers provide full compliance with this definition, support the lower case option, and extend the name to support up to 40 character names.

Data variables in IEC61131-3 may be defined such that they are accessible to all programs within a resource or controller, or limited access is provided only to the functions or routines within a single program. To pass data between multiple resources or controllers, access paths may be configured to define the location of the data within a system. Logix5000 controllers provide compliance by providing program scoped, controller scoped data and permits the configuration of access paths using produced/consumed data.

The memory interpretation of a variable within IEC61131-3 is defined through the use of either an elementary data type or an optional derived data type that is created from a group of multiple data types. Logix5000 controllers support the use of the BOOL (1 bit), SINT (8 bit integer), INT (16 bit integer), DINT (32 bit integer) and REAL (IEEE floating point number) elementary data types. Additionally, the optional derived data types are supported through the creation of user defined structures and arrays.

Programming Languages

The IEC61131-3 specification defines five (5) different programming languages and a set of common elements. All languages are defined as optional but at least one must be supported in order to claim compliance with the specification. The IEC61131-3 programming language components are defined as follows:

- Common Language Elements
- Common Graphical Elements
- Instruction List (IL) Language Elements
- Structured Text Language (ST) Elements
- Ladder Diagram (LD) Language Elements
- Sequential Function Chart (SFC) Language Elements
- Function Block Diagram (FBD) Language Elements

Logix5000 controllers and RSLogix5000 provide support for the common language elements and the Ladder Diagram and Function Block Diagram language options. Additionally, the environment utilizes an ASCII import/export format based on the Structured Text language. The instruction set and program file exchange features are discussed in detail in the sections that follow.

Instruction Set

The instruction set specified by IEC61131-3 is entirely optional. The specification lists a limited set of instructions that if implemented must conform to the stated execution and visual representation. IEC61131-3 however, does not limit the instructions set to those listed within the specification. Each PLC vendor is free to implement additional functionality in the form of instructions over and above those listed by the specification. Examples of such extended instructions are those needed to perform diagnostics, PID loop control, motion control and data file manipulation. Because extended instructions are not defined by the IEC61131-3 specification, there is no guarantee that the implementation between different PLC vendors will be compatible. Thus utilization of these instructions may preclude the movement of logic between vendors.

Logix5000 controllers and RSLogix5000 provide a suite of instructions that execute as defined by the IEC61131-3 specification. The physical representation of these instructions maintain their look and feel with existing systems so as to reduce the training cost associated with working with the environment. In addition to the IEC61131-3 compliant instructions, a full range of instructions from existing products have been brought forward into the environment so that no functionality is lost.

IEC61131-3 Program Portability

One of the goals of end-users creating programs in an IEC61131-3 compliant environment is the movement or portability of programs between controllers developed by different vendors. This area is a weakness of IEC61131-3 because no file exchange format is defined by the specification. This means that if any program created in one vendor's environment will require manipulation to move it to another vendor's system.

In order to minimize the effort involved in performing cross-vendor portability, the RSLogix 5000 programming software for the controllers includes a full ASCII export and import utility. Additionally, the file format that is utilized by this tool is based on a hybrid of the IEC61131-3 Structured Text language definition. Controller operating system and data definitions follow the appropriate IEC61131-3 formats. Extensions were implemented in order to convert Ladder Diagram logic into ASCII text since this is not defined by IEC61131-3.

For more information on the ASCII export and import utility of RSLogix 5000 programming software, see the *Logix5000 Controllers Import/Export Reference Manual*, publication 1756-RM084.

IEC Compliance Tables

Logix5000 controllers and RSLogix5000 comply with the requirements of IEC61131-3 for the following language features:

Table Number:[1]	Feature Number:	Feature Description:	Extensions and Implementation Notes:
1	2	Lower case letters	none
1	3a	Number sign (#)	Used for immediate value data type designation
1	4a	Dollar sign ($)	Used for description and string control character
1	6a	Subscript delimiters ([ ])	Array subscripts
2	1	Identifiers using upper case and numbers	Task, program, routine, structure and tag names
2	2	Identifiers using upper case, numbers, and embedded underlines	Task, program, routine, structure and tag names
2	3	Identifiers using upper and lower case, numbers and embedded underlines	Task, program, routine, structure and tag names
4	1	Integer literal	12, 0, -12
4	2	Real literal	12.5, -12.5
4	3	Real literal with exponents	$-1.34E^{-12}$, $1.234E^{6}$
4	4	Base 2 literal	2#0101_0101
4	5	Base 8 literal	8#377
4	6	Base 16 literal	16#FFE0
4	7	Boolean zero and one	0, 1
5	1	Empty String ''	Descriptions
5	2	String of length one containing a character 'A'	Descriptions
5	3	String of length one containing a space ' '	Descriptions
5	4	String of length one containing a single quote character '$'	Descriptions
6	2	String dollar sign '$$'	Descriptions
6	3	String single quote '$''	Descriptions
6	4	String Line Feed '$L' or '$l'	Descriptions
6	5	String New-line '$N' or '$n'	Descriptions
6	6	String From Feed (page) '$P' or '$p'	Descriptions
6	7	String Carriage return '$R' or '$r'	Descriptions
6	8	String Tab '$T' or '$t'	Descriptions
10	1	BOOL Data Type	Tag variable definition
10	2	SINT Data Type	Tag variable definition
10	3	INT Data Type	Tag variable definition
10	4	DINT Data Type	Tag variable definition
10	10	REAL Data Type	Tag variable definition
10	12	Time	Tag variable definition, TIMER Structure

Table Number:[1]	Feature Number:	Feature Description:	Extensions and Implementation Notes:
10	16	STRING data type	none
11	1	Data type Hierarchy	none
12	1	Direct Derivation from elementary types	User Defined data type structures
12	4	Array data types	Tag variable definition
12	5	Structured Data types	User defined data type structures
13	1	BOOL, SINT, INT, DINT initial value of 0	Tag variable definition
13	4	REAL, LREAL initial value of 0.0	Tag variable definition
13	5	Time initial value of T#0s	Tag variable definition, reset (RES) instruction
13	9	Empty String ''	Descriptions
14	1	Initialization of directly derived types	Import/export
14	4	Initialization of array data types	Import/export
14	5	Initialization of structured type elements	Import/export
14	6	Initialization of derived structured data types	Import/export
20	1	Use of EN and ENO	Function present in ladder but not labeled. Available in FBD.
20	2	Usage without EN and ENO	Available in FBD
20	3	Usage with EN and without ENO	Available in FBD
20	4	Usage without EN and with ENO	Available in FBD
21	1	Overloaded functions ADD(INT, DINT) or ADD(DINT, REAL)	All overloaded types that are supported are documented with each instruction
22	1	_TO_ conversion function	RAD, DEG instructions Radians to/from Decimal. Others not needed because of instruction overloading
22	2	Truncate conversion function	TRN instruction
22	3	BCD to INT Convert	FRD instruction
22	4	INT to BCD Convert	TOD instruction
23	1	Absolute value	ABS instruction
23	2	Square root	SQR instruction
23	3	Natural log	LN instruction
23	4	Log base 10	LOG instruction
23	6	Sine in radians	SIN instruction
23	7	Cosine in radians	COS instruction
23	8	Tangent in radians	TAN instruction
23	9	Principal arc sine	ASN instruction
23	10	Principal arc cosine	ACS instruction
23	11	Principal arc tangent	ATN instruction
24	12	Arithmetic add	ADD instruction

Table Number:[1]	Feature Number:	Feature Description:	Extensions and Implementation Notes:
24	13	Arithmetic multiplication	MUL instruction
24	14	Arithmetic subtraction	SUB instruction
24	15	Arithmetic divide	DIV instruction
24	16	Modulo	MOD instruction
24	17	Exponentiation	XPY instruction
24	18	Value move	MOV instruction in ladder
25	1	Bit shift left	Functionality contained in BSL instruction in Ladder for shift of 1
25	2	Bit shift right	Functionality contained in BSR instruction in Ladder for shift of 1
25	3	Bit rotate left	Functionality contained in BSL instruction in Ladder for shift of 1
25	4	Bit rotate right	Functionality contained in BSR instruction in Ladder for shift of 1
26	5	AND	BAND instruction in FBD
26	6	OR	BOR instruction in FBD
26	7	XOR	BXOR instruction in FBD
26	8	NOT	BNOT instruction in FBD
27	1	Select	SEL instruction in FBD
27	2a	Maximum select	Functionality contained in ESEL instruction in FBD
27	2b	Minimum select	Functionality contained in ESEL instruction in FBD
27	3	High/Low limit	HLL instruction in FBD
27	4	Multiplexer	MUX instruction in FBD
28	5	Comparison greater-than	GRT instruction
28	6	Comparison greater-than or equal	GRE instruction
28	7	Comparison equal	EQU instruction
28	8	Comparison less-than	LES instruction
28	9	Comparison less-than or equal	LEQ instruction
28	10	Comparison not equal	NEQ instruction
29	1	String length	Contained as parameter of STRING data type
29	4	Middle string	MID instruction in ladder
29	5	String concatenation	CONCAT instruction in ladder
29	6	String insert	INSERT instruction in ladder
29	7	String delete	DELETE instruction in ladder
29	9	Find string	FIND instruction in ladder
32	1	Input read	FBD editor

Table Number:[1]	Feature Number:	Feature Description:	Extensions and Implementation Notes:
32	2	Input write	FBD editor
32	3	Output read	FBD editor
32	4	Output write	FBD editor
34	1	Bistable set dominant	SETD instruction in FBD
34	2	Bistable reset dominant	RESD instruction in FBD
35	1	Rising edge detector	OSR instruction in ladder and OSRI instruction in FBD
35	2	Falling edge detector	OSF instruction in ladder and OSFI instruction in FBD
36	1b	Up-counter	Functionality contained in CTU and RES instructions in ladder and in CTUD instruction in FBD
37	2a	On-delay timer	Functionality contained in TON instruction in ladder and TONR instruction in FBD
37	3a	Off-delay timer	Functionality contained in TOF instruction in ladder and TOFR instruction in FBD
38	2	On-delay timing	Functionality contained in TON instruction in ladder and TONR instruction in FBD
38	3	Off-delay timing	Functionality contained in TOF instruction in ladder and TOFR instruction in FBD
57	1, 2	Horizontal line	Ladder editor, FBD editor
57	3, 4	Vertical line	Ladder editor, FBD editor
57	5, 6	Horizontal / Vertical connection	Ladder editor, FBD editor
57	9, 10	Connection and non-connection corners	Ladder editor, FBD editor
57	11, 12	Blocks with connections	Ladder editor, FBD editor
57	7,8	Line crossings without connection	FBD editor
57	13,14	Connectors	FBD editor
58	2	Unconditional jump	JMP instruction in ladder
58	3	Jump target	LBL instruction in ladder
58	4	Conditional jump	JMP instruction in ladder
58	5	Conditional return	RET instruction in ladder
58	8	Unconditional return	RET instruction in ladder
59	1	Left hand power rail	Ladder editor
59	2	Right hand power rail	Ladder editor
60	1	Horizontal link	Ladder editor
60	2	Vertical link	Ladder editor
61	1, 2	Normally open contact --| |--	XIC instruction in ladder
61	3, 4	Normally close contact --| / |--	XIO instruction in ladder
61	5, 6	Positive transition sensing contact -| P |-	ONS instruction in ladder

Table Number:[1]	Feature Number:	Feature Description:	Extensions and Implementation Notes:
62	1	Coil --( )--	OTE instruction in ladder
62	3	Set (latch) coil	Functionality contained in OTL instruction in ladder
62	4	Reset (unlatch) coil	Functionality contained in OTU instruction in ladder
62	8	Positive transition sensing coil	OSR instruction in ladder
62	9	Negative transition sensing coil	OSF instruction in ladder

[1]Table associated with languages other than ladder diagram and function block diagram have been skipped.

Notes:

A

alias tag

A tag that references another tag. An alias tag can refer to another alias tag or a base tag. An alias tag can also refer to a component of another tag by referencing a member of a structure, an array element, or a bit within a tag or member. See *base tag*.

ASCII

A 7-bit code (with an optional parity bit) that is used to represent alphanumerical characters, punctuation marks, and control-code characters. For a list of ASCII codes, see the back cover of this manual.

asynchronous

Actions that occur independent of each other and lack a regular pattern. In Logix5000 controllers, I/O values update asynchronous to the execution of logic.:

- Programs within a task access input and output data directly from controller-scoped memory.
- Logic within any task can modify controller-scoped data.
- Data and I/O values are asynchronous and can change during the course of a task's execution.
- An input value referenced at the beginning of a task's execution can be different when referenced later.

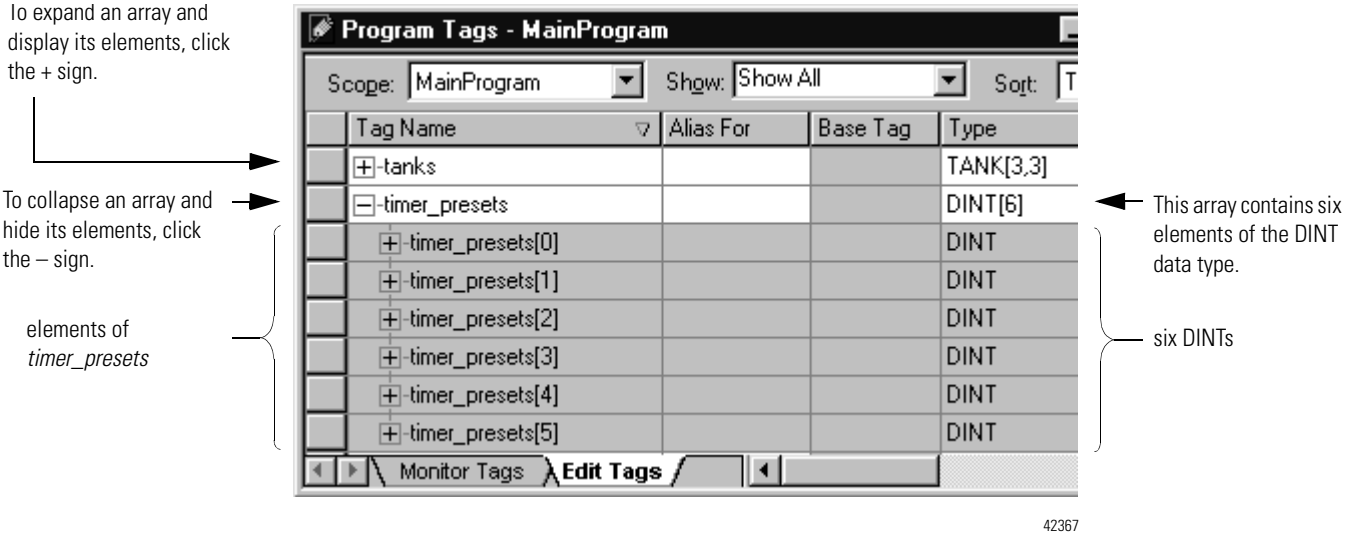
ATTENTION

Take care to ensure that data memory contains the appropriate values throughout a task's execution. You can duplicate or buffer data at the beginning of the scan to provide reference values for your logic.

array

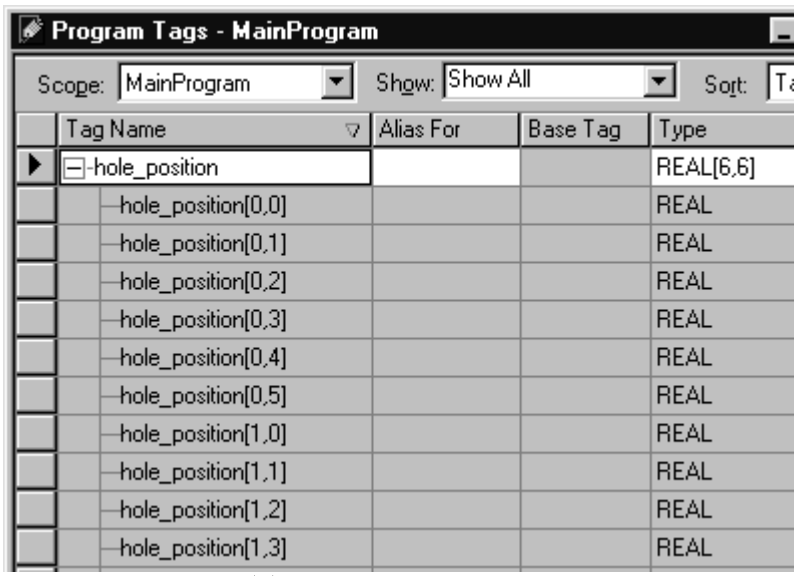
An array lets you group data (of the same data type) under a common name.

- An array is similar to a file.
- A subscript (s) identifies each individual **element** within the array.
- A subscript starts at 0 and extends to the number of elements minus 1 (zero based).



- An array tag occupies a contiguous block of memory in the controller, each element in sequence.
- You can use array and sequencer instructions to manipulate or index through the elements of an array
- An array can have as many as three dimensions. This gives you the flexibility to identify an element using one, two, or three subscripts (coordinates).

- In an array with two or three dimensions, the right-most dimension increments first in memory.

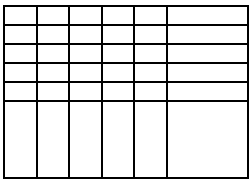
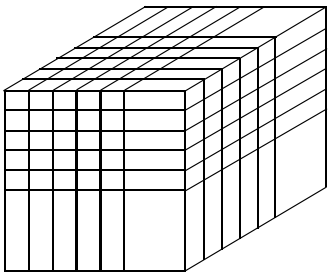


This array contains a two-dimensional grid of elements, six elements by six elements.

The right-most dimension increments to its maximum value then starts over.

When the right-most dimension starts over, the dimension to the left increments by one.

- The total number of elements in an array is the product of each dimension's size, as depicted in the following examples:

This array:	Stores data like:	For example:				
one dimension		Tag name:	Type	Dimension 0	Dimension 1	Dimension 2
		one_d_array	DINT[7]	7	--	--
		total number of elements = 7				
		valid subscript range DINT[x] where x=0–6				
two dimension		Tag name:	Type	Dimension 0	Dimension 1	Dimension 2
		two_d_array	DINT[4,5]	4	5	--
		total number of elements = 4 * 5 = 20				
		valid subscript range DINT[x,y] where x=0–3; y=0–4				
three dimension		Tag name:	Type	Dimension 0	Dimension 1	Dimension 2
		three_d_array	DINT[2,3,4]	2	3	4
		total number of elements = 2 * 3 * 4 = 24				
		valid subscript range DINT[x,y,z] where x=0–1; y=0–2, z=0–3				

- You can modify array dimensions when programming offline without loss of tag data. You cannot modify array dimensions when programming online.

application

The combination of routines, programs, tasks, and I/O configuration used to define the operation of a single controller. See *project*.

B

base tag

A tag that actually defines the memory where a data element is stored. See *alias tag*.

bidirectional connection

A connection in which data flows in both directions: from the originator to the receiver and from the receiver to the originator. See *connection, unidirectional connection*.

binary

Integer values displayed and entered in base 2 (each digit represents a single bit). Prefixed with 2#. Padded out to the length of the boolean or integer (1, 8, 16, or 32 bits). When displayed, every group of four digits is separated by an underscore for legibility. See *decimal*, *hexadecimal*, *octal*.

bit

Binary digit. The smallest unit of memory. Represented by the digits 0 (cleared) and 1 (set).

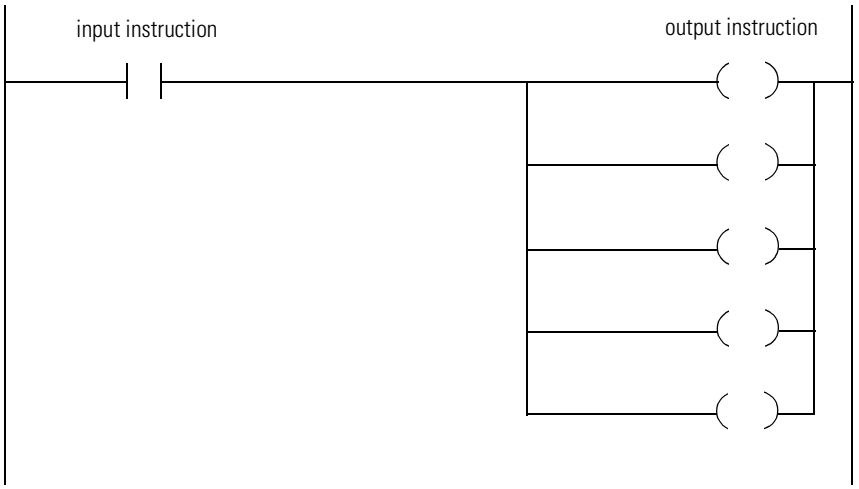
BOOL

An data type that stores the state of a single bit, where:

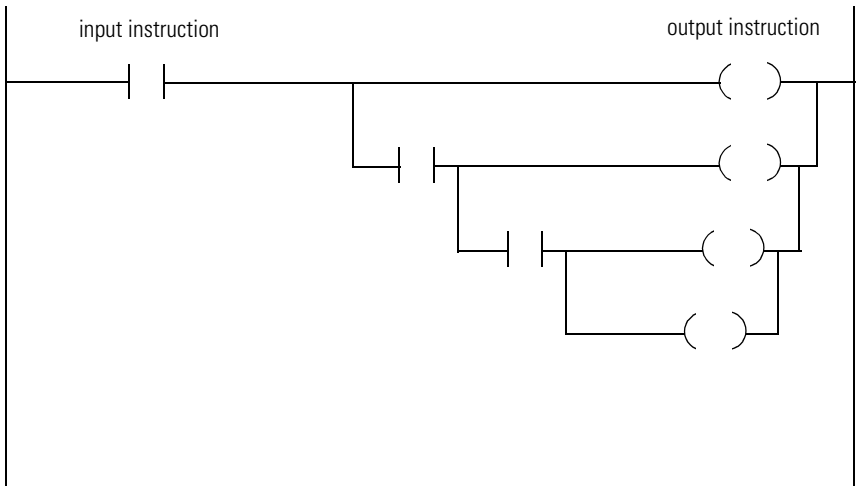
- *0* equals *off*
- *1* equals *on*

branch

There is no limit to the number of parallel branch levels that you can enter. The following figure shows a parallel branch with five levels. The main rung is the first branch level, followed by four additional branches.



You can nest branches to as many as 6 levels. The following figure shows a nested branch. The bottom output instruction is on a nested branch that is three levels deep.



byte

A unit of memory consisting of 8 bits.

C

cached connection

With the MSG instruction, a cached connection instructs the controller to maintain the connection even after the MSG instruction completes. This is useful if you repeatedly execute the MSG instruction because initiating the connection each time increases scan time. See *connection, uncached connection*.

change of state (COS)

Any change in the status of a point or group of points on an I/O module.

CIP

See Control and Information Protocol.

communication format

Defines how an I/O module communicates with the controller. Choosing a communication format defines:

- what configuration tabs are available through the programming software
- the tag structure and configuration method

compatible module

An electronic keying protection mode that requires that the vendor, catalog number, and major revision attributes of the physical module and the module configured in the software match in order to establish a connection to the module. See *disable keying*, *exact match*.

connection

The communication mechanism from the controller to another module in the control system. The number of connections that a single controller can have is limited. Communications with I/O modules, consumed tags, produced tags, and MSG instructions use connections to transfer data.

consumed tag

A tag that receives the data that is broadcast by a produced tag over a ControlNet network or ControlLogix backplane. A consumed tag must be:

- controller scope
- same data type (including any array dimensions) as the remote tag (produced tag)

See *produced tag*.

continuous task

The task that runs continuously.

- The continuous task runs in the background. Any CPU time not allocated to other operations (such as motion, communications, and periodic tasks) is used to execute the programs within the continuous task.
- The continuous task restarts itself after the last of its programs finishes.
- A project does not require a continuous task.
- If used, there can be only one continuous task.
- All periodic tasks interrupt the continuous task.
- When you create a project, the default *MainTask* is the continuous task. You can leave this task as it is, or you can change its properties (name, type, etc.).

See *periodic task*.

Control and Information Protocol

Messaging protocol used by Allen-Bradley's Logix5000 line of control equipment. Native communications protocol used on the ControlNet network.

controller fault handler

The controller fault handler is an optional task that executes when the:

- major fault is not an instruction-execution fault
- program fault routine:
  - could not clear the major fault
  - faulted
  - does not exist

You can create only one program for the controller fault handler. After you create that program, you must configure one routine as the main routine.

- The controller fault program does *not* execute a fault routine.
- If you specify a fault routine for the controller fault program, the controller never executes that routine.
- You can create additional routines and call them from the main routine.

controller scope

Data accessible anywhere in the controller. The controller contains a collection of tags that can be referenced by the routines and alias tags in any program, as well as other aliases in the controller scope.
See *program scope*.

Coordinated System Time (CST)

A 64-bit value that represents the number of microseconds since the CST master controller started counting.

- The CST value is stored as a DINT[2] array, where:
  - first element stores the lower 32 bits
  - second element stores the upper 32 bits
- You can use the CST timestamp to compare the relative time between data samples.

COUNTER

Structure data type that contains status and control information for counter instructions

D

data type

A definition of the memory size and layout that will be allocated when you create a tag of that data type.

decimal

Integer values displayed and entered in base 10. No prefix. Not padded to the length of the integer. See *binary*, *hexadecimal*, *octal*.

description

Optional text that you can use to further document your application.

- You can use any printable character, including carriage return, tab, and space.
- Descriptions do not download to the controller. They remain in the offline project file.
- Descriptions have these length limitations:
  - For tags, you can use up to 120 characters.
  - For other objects (tasks, programs, modules, etc.), you can use up to 128 characters.

dimension

Specification of the size of an array. Arrays can have as many as three dimensions. See *array*.

DINT

A data type that stores a 32-bit (4-byte) signed integer value (-2,147,483,648 to +2,147,483,647). In Logix5000 controllers, use DINTs for integers:

- Logix5000 controllers execute more efficiently and use less memory when working with 32-bit integers (DINTs) instead of 16-bit integers (INTs) or 8-bit integers (SINTs).
- Typically, instructions convert SINT or INT values to an **optimal data type** (usually a DINT or REAL value) during execution. Because this requires additional time and memory, minimize the use of the SINT and INT data types.

direct

An I/O connection where the controller establishes an individual connections with an I/O module. See *rack optimized*.

disable keying

An electronic keying protection mode that requires no attributes of the physical module and the module configured in the software to match and still establishes a connection to the module. See *compatible module*, *exact match*.

download

The process of transferring the contents of a project on the workstation into the controller. See *upload*.

E

elapsed time

The total time required for the execution of all operations configured within a single task.

- If the controller is configured to run multiple tasks, elapsed time includes any time used/shared by other tasks performing other operations.
- While online, you can use the *Task Properties* dialog box to view the maximum scan time and the last scan time in ms for the current task. These values are elapsed time, which includes any time spent waiting for higher-priority tasks.

See *execution time*.

electronic keying

A feature of the 1756 I/O line where modules can be requested to perform an electronic check to insure that the physical module is consistent with what was configured by the software. Enables the user via the software to prevent incorrect modules or incorrect revisions of modules from being inadvertently used. See *compatible module*, *disable keying*, *exact match*.

element

An addressable unit of data that is a sub-unit of a larger unit of data. A single unit of an array.

- You specify an element in an array by its subscript(s):

For this array:	Specify:
one dimension	*array_name[subscript_0]*
two dimension	*array_name[subscript_0, subscript_1]*
three dimension	*array_name[subscript_0, subscript_1, subscript_2]*

See *array*.

exact match

An electronic keying protection mode that requires that all attributes (vendor, catalog number, major revision, and minor revision) of the physical module and the module configured in the software match in order to establish a connection to the module.

execution time

The total time required for the execution of a single program.

- Execution time includes only the time used by that single program, and excludes any time shared/used by programs in other tasks performing other operations.
- When online, use the *Program Properties* dialog box to view the maximum scan time and the last scan time (in µs) for the current program. These values are execution times for the program and do not include any time spent waiting for other programs or higher-priority tasks.

See *elapsed time*.

exponential

Real values displayed and entered in scientific or exponential format. The number is always displayed with one digit to the left of the decimal point, followed by the decimal portion, and then by an exponent. See *style*.

F

faulted mode

The controller generated a major fault, could not clear the fault, and has shut down.

See *major fault*.

float

Real values displayed and entered in floating point format. The number of digits to the left of the decimal point varies according to the magnitude of the number. See *style*.

H

hexadecimal

Integer values displayed and entered in base 16 (each digit represents four bits). Prefixed with 16#. Padded out to length of the boolean or integer (1, 8, 16, or 32 bits). When displayed, every group of four digits is separated by an underscore for legibility. See *binary*, *decimal*, *octal*.

I

immediate value

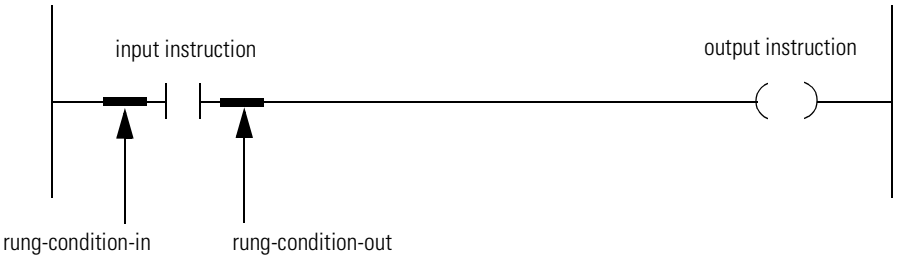
An actual 32-bit signed real or integer value. Not a tag that stores a value.

index

A reference used to specify an element within an array.

instruction

The controller evaluates ladder instructions based on the rung condition preceding the instruction (rung-condition-in).



Only input instructions affect the rung-condition-in of subsequent instructions on the rung:

- If the rung-condition-in to an input instruction is true, the controller evaluates the instruction and sets the rung-condition-out to match the results of the evaluation.
  - If the instruction evaluates to true, the rung-condition-out is true.
  - If the instruction evaluates to false, the rung-condition-out is false.
- An output instruction does not change the rung-condition-out.
  - If the rung-condition-in to an output instruction is true, the rung-condition-out is set to true.
  - If the rung-condition-in to an output instruction is false, the rung-condition-out is set to false.

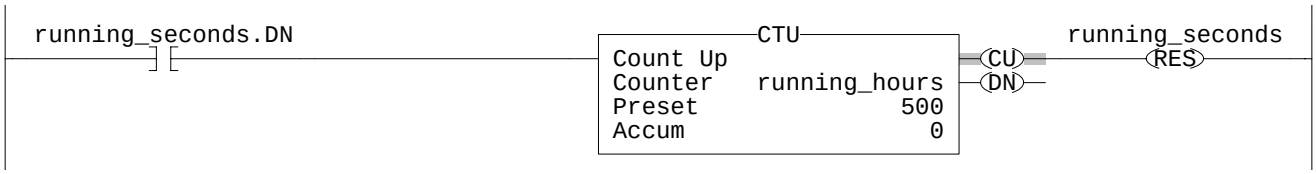
In Logix5000 controllers, you can enter multiple output instructions per rung of logic. You can enter the output instructions:

- in sequence on the rung (serial)
- between input instructions, as long as the last instruction on the rung is an output instruction

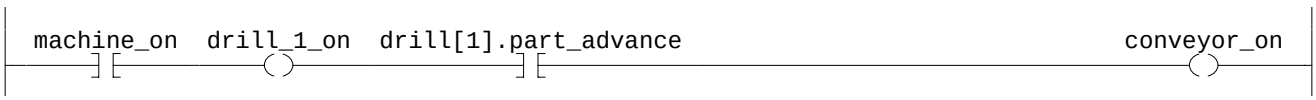
The following example uses more than one output on a rung.

EXAMPLE Place multiple outputs on a rung

When *running_seconds.DN* turns on, *running_hours* counts up by one and *running_seconds* resets.



When *machine_on* turns on, turns on *drill_1_on*. When both *machine_on* and *drill[1].part_advance* are on, turns on *conveyor_on*.



42362

INT

A data type that stores a 16-bit (2-byte) integer value (-32,768 to +32,767). Minimize your use of this data type:

- Typically, instructions convert SINT or INT values to an **optimal data type** (usually a DINT or REAL value) during execution. Because this requires additional time and memory, minimize the use of the SINT and INT data types.

interface module (IFM)

A pre-wired I/O field wiring arm.

L

listen-only connection

An I/O connection where another controller owns/provides the configuration data for the I/O module. A controller using a listen-only connection does not write configuration data and can only maintain a connection to the I/O module when the owner controller is actively controlling the I/O module. See *owner controller*.

load

To copy a project from nonvolatile memory to the user memory (RAM) of the controller. This overwrites any project that is currently in the controller. See *nonvolatile memory, store*.

M

main routine

The first routine to execute when a program executes. Use the main routine to call (execute) other routines (subroutines).

major fault

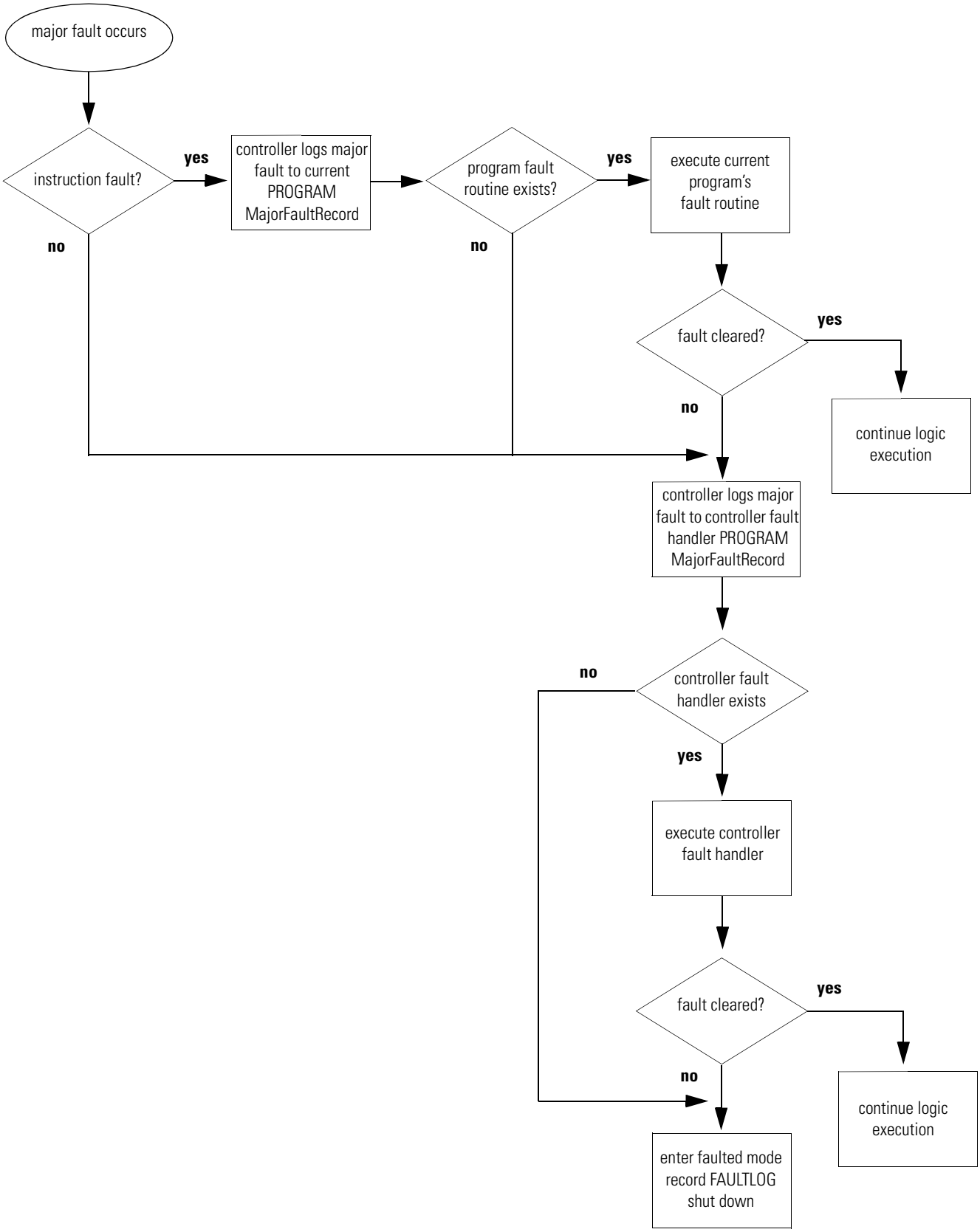
A fault condition that is severe enough for the controller to shut down, unless the condition is cleared. When a major fault occurs, the controller:

1. Sets a major fault bit
2. Runs user-supplied fault logic, if it exists
3. If the user-supplied fault logic cannot clear the fault, the controller goes to faulted mode
4. Sets outputs according to their output state during program mode
5. OK LED flashes red

The controller supports two levels for handling major faults:

- program fault routine:
  - Each program can have its own fault routine.
  - The controller executes the program's fault routine when an instruction fault occurs.
  - If the program's fault routine does not clear the fault or a program fault routine does not exist, the controller proceeds to execute the controller fault handler (if defined).
- controller fault handler:
  - If the controller fault handler does not exist or cannot clear the major fault, the controller enters faulted mode and shuts down. At this point, the FAULTLOG is updated. (See the next page.)
  - All non-instruction faults (I/O, task watchdog, etc.) execute the controller fault handler directly. (No program fault routine is called.)

The fault that was not cleared, and up to two additional faults that have not been cleared, are logged in the controller fault log.


major fault occurs
instruction fault?
yes
no
controller logs major fault to current PROGRAM MajorFaultRecord
program fault routine exists?
yes
no
execute current program's fault routine
fault cleared?
yes
no
continue logic execution
controller logs major fault to controller fault handler PROGRAM MajorFaultRecord
controller fault handler exists
no
yes
execute controller fault handler
fault cleared?
yes
no
continue logic execution
enter faulted mode record FAULTLOG shut down

See faulted state, minor fault.

major revision

The 1756 line of modules have major and minor revision indicators. The major revision is updated any time there is a functional change to the module. See *electronic keying*, *minor revision*.

master (CST)

Within a single chassis, one and only one, controller must be designated as the Coordinated System Time (CST) master. All other modules in the chassis synchronize their CST values to the CST master.

member

An element of a structure that has its own data type and name.

- Members can be structures as well, creating nested structure data types.
- Each member within a structure can be a different data type.
- To reference a member in a structure, use this format:

  ```
  tag_name.member_name
  ```

  For example:

This address:	References the:
timer_1.pre	PRE value of the *timer_1* structure.
input_load as data type *load_info* *input_load.height*	*height* member of the user-defined *input_load* structure

- If the structure is embedded in another structure, use the tag name of the structure at the highest level followed by a substructure tag name and member name:

  ```
  tag_name.substructure_name.member_name
  ```

  For example:

This address:	References the:
input_location as data type *location* *input_location.load_info.height*	*height* member of the *load_info* structure in the *input_location* structure.

- If the structure defines an array, use the array tag, followed by the position in the array and any substructure and member names.

  ```
  array_tag[position].member
  ```

  or

  ```
  array_tag[position].substructure_name.member
  _name
  ```

  For example:

This address:	References the:
conveyor[10].source	*source* member of the 11th element in the *conveyor* array (array elements are zero based).
conveyor[10].info.height	*height* member of the *info* structure in the 11th element of the *conveyor* array (array elements are zero based).

See *structure*.

memory

Electronic storage media built into a controller, used to hold programs and data.

minor fault

A fault condition that is *not* severe enough for the controller to shut down:

If this occurs:	The controller:
problem with an instruction	1. sets S:MINOR 2. logs information about the fault to the PROGRAM object, MinorFaultRecord attribute 3. sets bit 4 of the FAULTLOG object, MinorFaultBits attribute
periodic task overlap	sets bit 6 of the FAULTLOG object, MinorFaultBits attribute
problem with the serial port	sets bit 9 of the FAULTLOG object, MinorFaultBits attribute
low battery	sets bit 10 of the FAULTLOG object, MinorFaultBits attribute

To clear minor faults:

1. In the controller organizer, right-click the *Controller* `name_of_controller` folder and select *Properties*.
2. Click the *Minor Fault*s tab.
3. Use the information in the *Recent Faults* list to correct the cause of the fault. Refer to "Minor Fault Codes" on page A-3.
4. Click the *Clear Minor*s button.

See *major fault*.

minor revision

The 1756 line of modules have major and minor revision indicators. The minor revision is updated any time there is a change to a module that does not affect its function or interface. See *electronic keying*, *major revision*.

multicast

A mechanism where a module can send data on a network that is simultaneously received by more that one listener. Describes the feature of the ControlLogix I/O line which supports multiple controllers receiving input data from the same I/O module at the same time.

multiple owners

A configuration setup where more than one controller has exactly the same configuration information to simultaneously own the same input module.

N

name

Names identify controllers, tasks, programs, tags, modules, etc. Names follow IEC-1131-3 identifier rules and:

- must begin with an alphabetic character (A-Z or a-z) or an underscore (_)
- can contain only alphabetic characters, numeric characters, and underscores
- can have as many as 40 characters
- must not have consecutive or trailing underscore characters (_)
- are *not* case sensitive
- download to the controller

network update time (NUT)

The repetitive time interval in which data can be sent on a ControlNet network. The network update time ranges from 2ms-100ms.

nonvolatile memory

Memory of the controller that retains its contents while the controller is without power or a battery. See *load*, *store*.

O

object

A structure of data that stores status information. When you enter a GSV/SSV instruction, you specify the object and its attribute that you want to access. In some cases, there are more than one instance of the same type of object, so you might also have to specify the object name. For example, there can be several tasks in your application. Each task has its own TASK object that you access by the task name.

octal

Integer values displayed and entered in base 8 (each digit represents three bits). Prefixed with 8#. Padded out to the length of the boolean or integer (1, 8, 16, or 32 bits). When displayed, every group of three digits is separated by an underscore for legibility. See *binary*, *decimal*, *hexadecimal*.

offline

Viewing and editing a project that is on the hard disk of a workstation. See *online*.

online

Viewing and editing the project in a controller. See *offline*.

optimal data type

A data type that a Logix5000 instruction actually uses (typically the DINT and REAL data types).

- In the instruction set reference manuals, a **bold** data type indicates an optimal data type.
- Instructions execute faster and require less memory if all the operands of the instruction use:
  - the same data type
  - an optimal data type
- If you mix data types and use tags that are not the optimal data type, the controller converts the data according to these rules
  - Are *any* of the operands a REAL value?

If:	Then input operands (e.g., source, tag in an expression, limit) convert to:
Yes	REALs
No	DINTs

  - After instruction execution, the result (a DINT or REAL value) converts to the destination data type, if necessary.
- Because the conversion of data takes additional time and memory, you can increase the efficiency of your programs by:
  - using the same data type throughout the instruction
  - minimizing the use of the SINT or INT data types

  In other words, use all DINT tags or all REAL tags, along with immediate values, in your instructions.

- The following table summarizes how the controller converts data between data types:

Conversion:	**Result:**		
larger integer to smaller integer	The controller truncates the upper portion of the larger integer and generates an overflow. For example:		
		Decimal	Binary
	DINT	65,665	0000_0000_0000_0001_0000_0000_1000_0001
	INT	129	0000_0000_1000_0001
	SINT	-127	1000_0001
SINT or INT to REAL	No data precision is lost		
DINT to REAL	Data precision could be lost. Both data types store data in 32 bits, but the REAL type uses some of its 32 bits to store the exponent value. If precision is lost, the controller takes it from the least-significant portion of the DINT.		
REAL to integer	The controller rounds the fractional part and truncates the upper portion of the non-fractional part. If data is lost, the controller sets the overflow status flag. Numbers round as follows: • Numbers other than *x*.5 round to the nearest number. • *X*.5 rounds to the nearest even number. For example:		
	REAL (source)	DINT (result)	
	-2.5	-2	
	-1.6	-2	
	-1.5	-2	
	-1.4	-1	
	1.4	1	
	1.5	2	
	1.6	2	
	2.5	2	

owner controller

The controller that creates the primary configuration and communication connection to a module. The owner controller writes configuration data and can establish a connection to the module. See *listen-only connection*.

P

path

A description of the devices and networks between one device and another. A connection from one device to another follows the specified path. See *connection*.

periodic task

A task that is triggered by the operating system at a repetitive period of time.

- Use a periodic task for functions that require accurate or deterministic execution.
- Whenever the time expires, the task is triggered and its programs are executed.
- Data and outputs established by the programs in the task retain their values until the next execution of the task or they are manipulated by another task.
- You can configure the time period from 1 ms to 2000 s. The default is 10 ms.

ATTENTION

Ensure that the time period is longer than the sum of the execution times of all the programs assigned to the task. If the controller detects that a periodic task trigger occurs for a task that is already operating, a minor fault occurs.

- Periodic tasks always interrupt the continuous task.
- Depending on the priority level, a periodic task may interrupt other periodic tasks in the controller.

See *continuous task*.

periodic task overlap

A condition that occurs when a task is executing and the same task is triggered again. The execution time of the task is greater than the periodic rate configured for the task. See *periodic task*.

predefined structure

A structure data type that stores related information for a specific instruction, such as the TIMER structure for timer instructions. Predefined structures are always available, regardless of the system hardware configuration. See *product defined structure*.

prescan

A function of the controller where the logic is examined prior to execution in order initialize instructions and data.

The controller performs prescan when you change the controller from Program mode to Run mode.

priority

Specifies which task to execute first if two tasks are triggered at the same time.

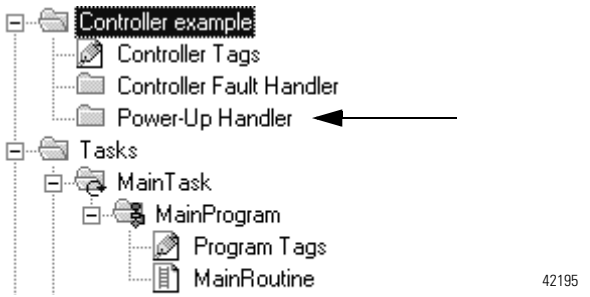
- The task with the higher priority executes first.
- Priorities range from 1-15, with 1 being the highest priority.
- A higher priority task will interrupt any lower priority task.
- If two tasks with the same priority are triggered at the same time, the controller switches between the tasks every millisecond.

postscan

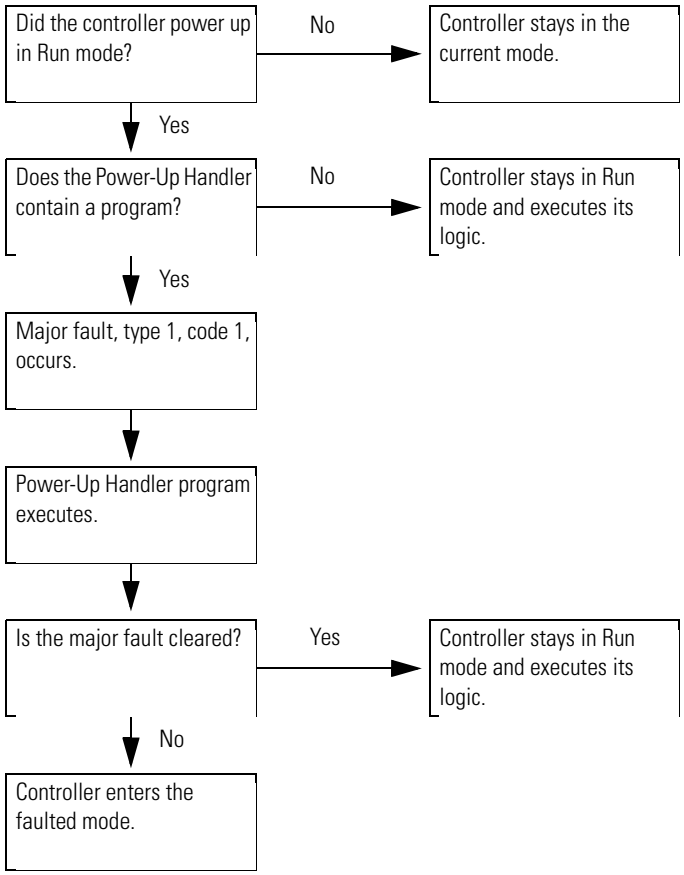
A function of the controller where the logic within a program is examined before disabling the program in order reset instructions and data.

power-up handler

An optional task that executes when the controller powers up in the Run mode. To use the Power-Up Handler, you must create a power-up program and associated main routine.



The Power-Up Handler executes as follows:


Did the controller power up in Run mode?
No
Controller stays in the current mode.
Yes
Does the Power-Up Handler contain a program?
No
Controller stays in Run mode and executes its logic.
Yes
Major fault, type 1, code 1, occurs.
Power-Up Handler program executes.
Is the major fault cleared?
Yes
Controller stays in Run mode and executes its logic.
No
Controller enters the faulted mode.

produced tag

A tag that a controller is making available for use by other controllers. Produced tags are always at controller scope. See *consumed tag*.

product defined structure

A structure data type that is automatically defined by the software and controller. By configuring an I/O module you add the product defined structure for that module.

program

A set of related routines and tags.

- Each program contains program tags, a main executable routine, other routines, and an optional fault routine.
- To execute the routines in a program, you assign (schedule) the program to a task:
  - When a task is triggered, the scheduled programs within the task execute to completion from first to last.
  - When a task executes a program, the main routine of the program executes first.
  - The main routine can, in turn, execute subroutines using the JSR instruction.
- The *Unscheduled Program*s folder contains programs that aren't assigned to a task.
- If the logic in the program produces a major fault, execution jumps to a configured fault routine for the program.
- The routines within a program can access the following tags:
  - program tags of the program
  - controller tags
- Routines cannot access the program tags of other programs.

See *routine*, *task*.

program scope

Data accessible only within the current program. Each program contains a collection of tags that can only be referenced by the routines and alias tags in that program. See *controller scope*.

project file

The file on your workstation (or server) that stores the logic, configuration, data, and documentation for a controller.

- The project file has an .ACD extension.
- When you create a project file, the file name is the name of the controller.
- The controller name is independent of the project file name. If you save a current project file as another name, the controller name is unchanged.
- If the name of the controller is different than the name of the project file, the title bar of the RSLogix 5000 software displays both names.

See *application*.

R

rack optimized

An I/O connection where the 1756-CNB module collects digital I/O words into a rack image (similar to 1771-ASB). A rack optimized connection conserves ControlNet connections and bandwidth, however, limited status and diagnostic information is available when using this connection type. See *direct*.

rate

For a periodic task, the rate at which the controller executes the task, from 1 ms to 2,000,000 ms (2000 seconds). The default is 10 ms.

REAL

A data type that stores a 32-bit (4-byte) IEEE floating-point value, with the following range:

- $-3.402823E^{38}$ to $-1.1754944E^{-38}$ (negative values)
- 0
- $1.1754944E^{-38}$ to $3.402823E^{38}$ (positive values)

The REAL data type also stores ±infinity and ±NAN, but the software display differs based on the display format.

Display Format:	Equivalent:	
Real	+infinite - infinite +NAN -NAN	1.$ -1.$ 1.#QNAN -1.#QNAN
Exponential	+infinite - infinite +NAN -NAN	1.#INF000e+000 -1.#INF000e+000 1.#QNAN00e+000 -1.#QNAN00e+000

removal and insertion under power (RIUP)

A ControlLogix feature that allows a user to install or remove a module while chassis power is applied.

requested packet interval (RPI)

When communicating over a the network, this is the maximum amount of time between subsequent production of input data.

- Typically, this interval is configured in microseconds.
- The actual production of data is constrained to the largest multiple of the network update time that is smaller than the selected RPI.
- Use a power of two times the ControlNet network update time (NUT).

  For example, if the NUT is 5 ms, type a rate of 5, 10, 20, 40 ms, etc.

See *network update time (NUT)*.

routine

A set of logic instructions in a single programming language, such as a ladder diagram.

- Routines provide the executable code for the project in a controller (similar to a program file in a PLC or SLC controller).
- Each program has a main routine:
  - When the controller triggers the associated task and executes the associated program, the main routine is the first routine to execute.
  - To call another routine within the program, enter a JSR instruction in the main routine.
- You can also specify an optional program fault routine.
  - If any of the routines in the associated program produce a major fault, the controller executes program fault routine

See *program*, *task*.

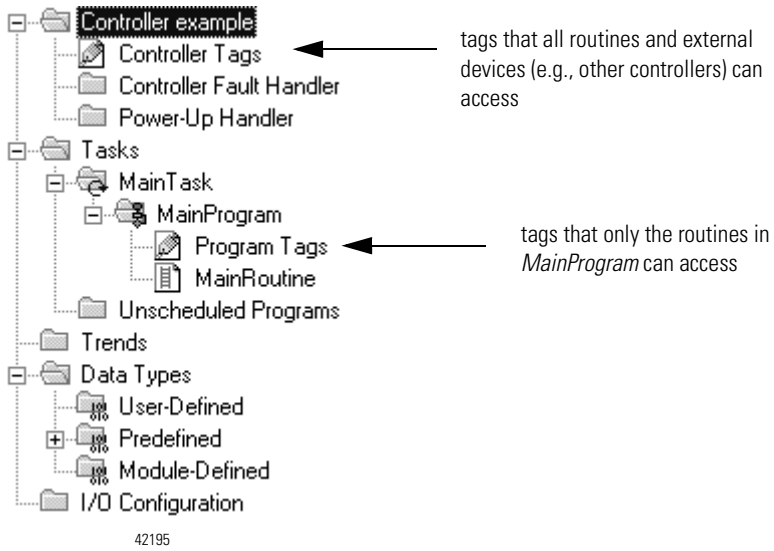
S

scan time

See *elapsed time*, *execution time*.

scope

Defines where you can access a particular set of tags. When you create a tag, you assign (scope) it as either a controller tag or a program tag for a specific program, as depicted below.



You can have multiple tags with the same name:

- Each tag must have a different scope. For example, one of the tags can be a controller tag and the other tags can be program tags for different programs. Or, each tag can be a program tag for a different program.
- Within a program, you cannot reference a controller tag if a tag of the same name exists as a program tag for that program.

See *controller scope*, *program scope*.

SINT

A data type that stores an 8-bit (1-byte) signed integer value (-128 to +127). Minimize your use of this data type:

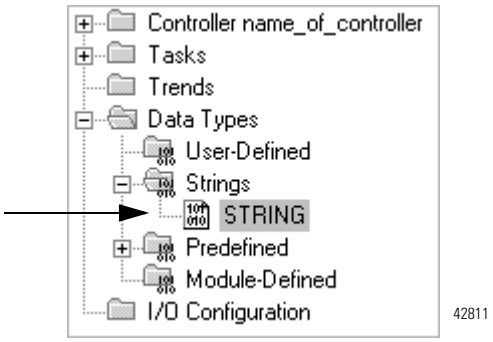
- Typically, instructions convert SINT or INT values to an **optimal data type** (usually a DINT or REAL value) during execution. Because this requires additional time and memory, minimize the use of the SINT and INT data types.

store

To copy a project to the nonvolatile memory of the controller. This overwrites any project that is currently in the nonvolatile memory. See *load, nonvolatile memory*.

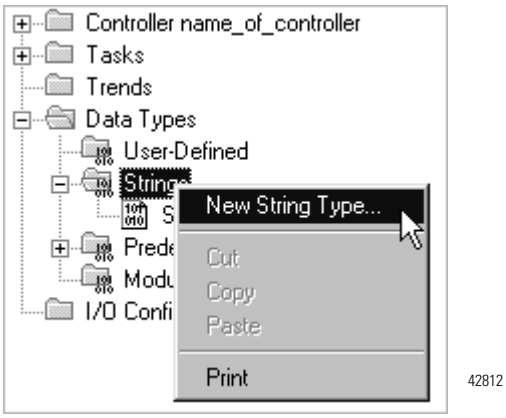
string

A group of data types that store ASCII characters.



You can use the default STRING data type. It stores up to 82 characters.

or



You can create a new string data type to store the number of characters that you define.

Each string data type contains the following members:

Name:	Data Type:	Description:	Notes:
LEN	DINT	number of characters in the string	The LEN automatically updates to the new count of characters whenever you: • use the String Browser dialog box to enter characters • use instructions that read, convert, or manipulate a string The LEN shows the length of the current string. The DATA member may contain additional, old characters, which are not included in the LEN count.
DATA	SINT array	ASCII characters of the string	• To access the characters of the string, address the name of the tag. For example, to access the characters of the *string_1* tag, enter *string_1*. • Each element of the DATA array contains one character. • You can create new string data types that store less or more characters.

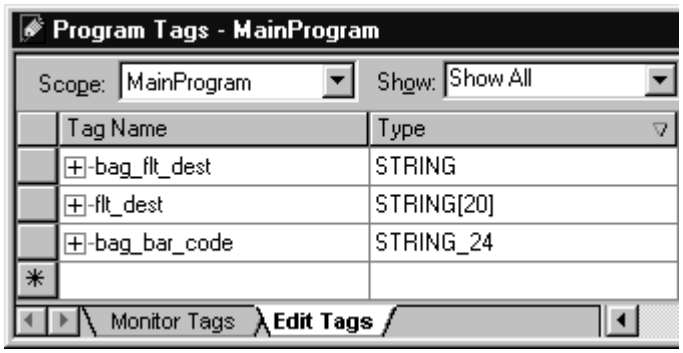
New string data types are useful in the following situations:

- If you have a large number of strings with a fixed size that is less than 82 characters, you can conserve memory by creating a new string data type.
- If you must handle strings that have more than 82 characters, you can create a new string data type to fit the required number of characters.

IMPORTANT Use caution when you create a new string data type. If you later decide to change the size of the string data type, you may lose data in any tags that currently use that data type.

If you:	Then:
make a string data type smaller	• The data is truncated. • The LEN is unchanged.
make a string data type larger	The data and LEN is reset to zero.

The following example shows the STRING data type and a new string data type.



This tag uses the default STRING data type.

This tag is an 20 element array of the default STRING data type.

This tag uses a new string data type.

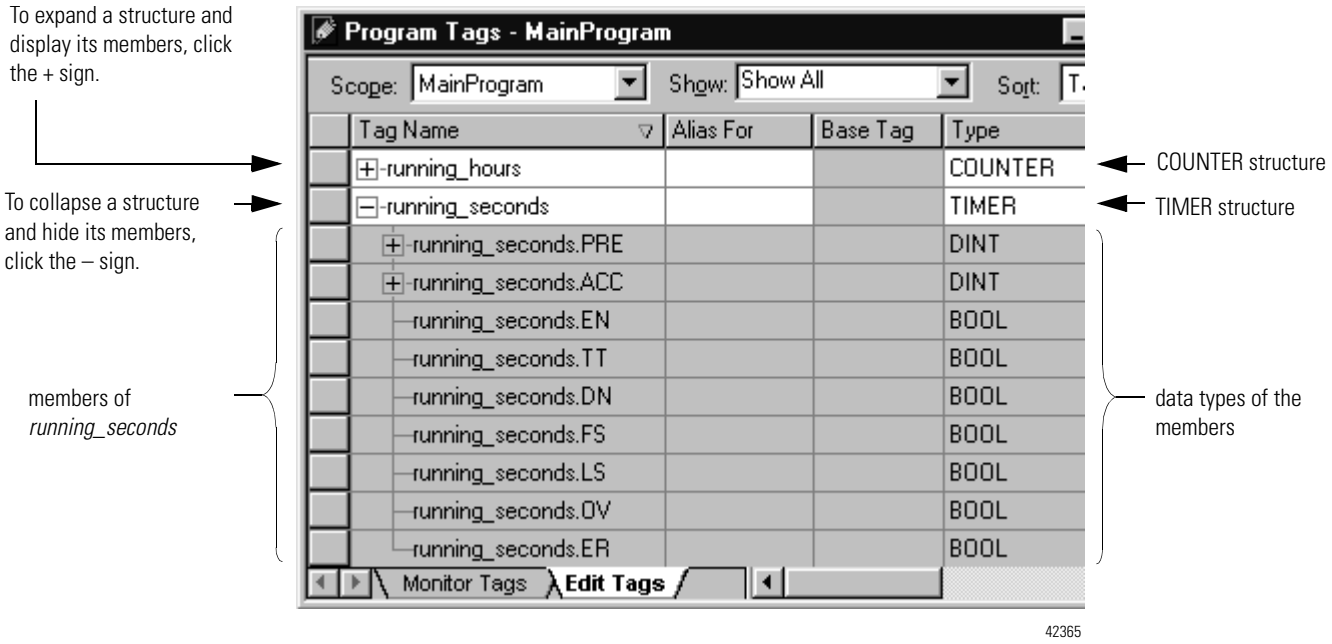
- The user named the string data type *STRING_24*.
- The new string data type stores only 24 characters.

structure

Some data types are a structure.

- A structure stores a group of data, each of which can be a different data type.
- Within a structure, each individual data type is called a **member**.
- Like tags, members have a name and data type.
- You create your own structures, called a **user-defined data type**, using any combination of individual tags and most other structures.
- To copy data to a structure, use the COP instruction. See the *Logix5000 Controllers General Instruction Set Reference Manual*, publication 1756-RM003.

The COUNTER and TIMER data types are examples of commonly used structures.



See *member*, *user-defined data type.*

style

The format that numeric values are displayed in. See *ASCII*, *binary, decimal, exponential, float, hexadecimal, octal.*

system overhead time slice

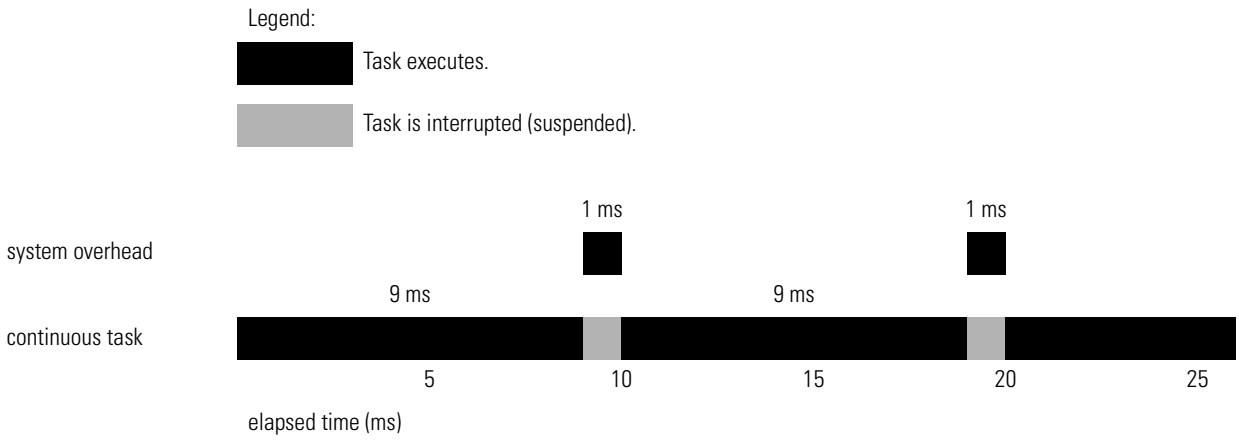
Specifies the percentage of controller time (excluding the time for periodic tasks) that is devoted to communication and background functions (system overhead):

- The controller performs system overhead functions for up to 1 ms at a time.
- If the controller completes the overhead functions in less than 1 ms, it resumes the continuous task.
- Communication and background functions include the following:
  - communicate with programming and HMI devices (such as RSLogix 5000 software)
  - respond to messages
  - send messages, including block-transfers
  - re-establish and monitor I/O connections (such as RIUP conditions); this *does not* include normal I/O communications that occur during program execution
  - bridge communications from the serial port of the controller to other ControlLogix devices via the ControlLogix backplane
- If communications are not completing fast enough, increase the system overhead timeslice.

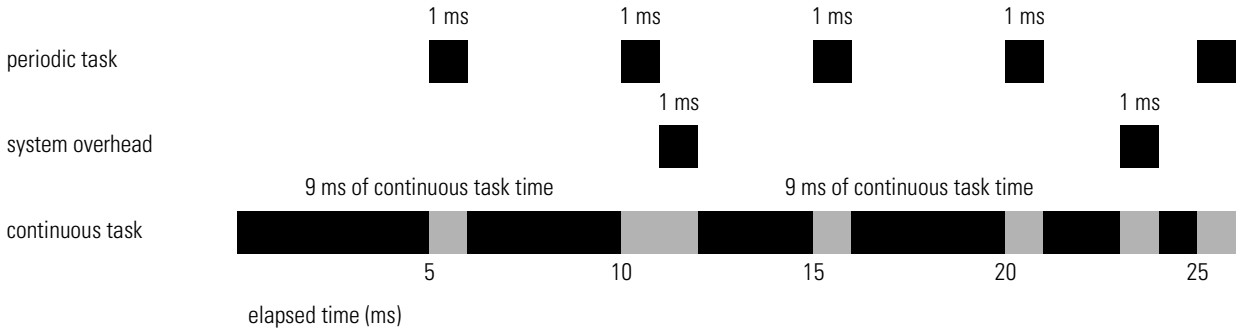
The following table shows the ratio between the continuos task and the system overhead functions:

At this time slice:	The continuous tasks runs for:	And then overhead occurs for up to:
10%	9 ms	1 ms
20%	4 ms	1 ms
33%	2 ms	1 ms
50%	1 ms	1 ms

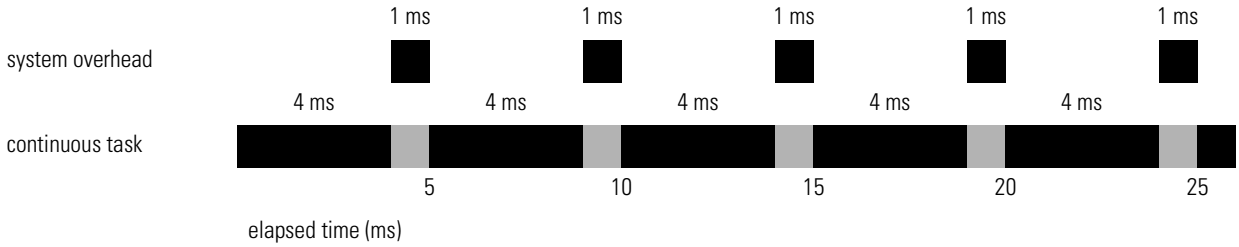
At the default time slice of 10 %, system overhead interrupts the continuous task every 9 ms (of continuous task time).



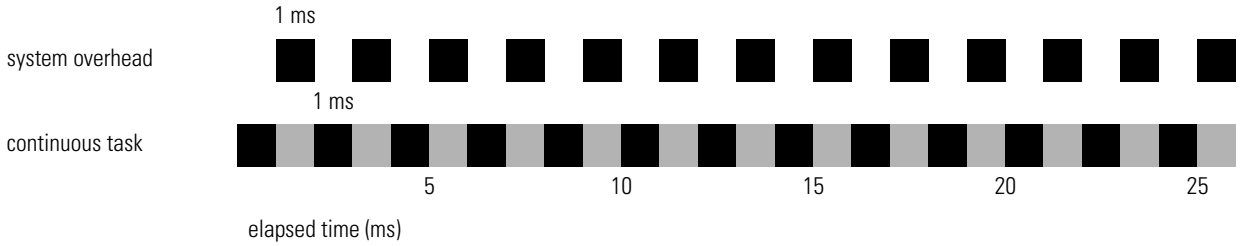
The interruption of a periodic task increases the elapsed time (clock time) between the execution of system overhead.



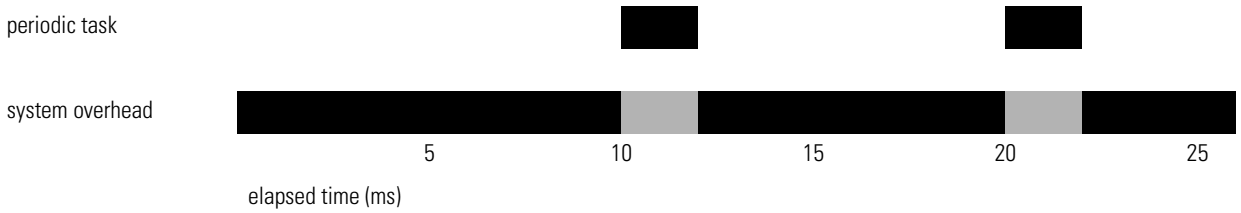
If you increase the time slice to 20 %, the system overhead interrupts the continuous task every 4 ms (of continuous task time).



If you increase the time slice to 50 %, the system overhead interrupts the continuous task every 1 ms (of continuous task time).



If the controller only contains a periodic task (s), the system overhead timeslice value has no effect. System overhead runs whenever a periodic task is not running.



To change the system overhead time slice:

1. Open the RSLogix 5000 project.
2. In the controller organizer, right-click the *Controller* `name_of_controller` folder and select *Properties*.
3. Click the *Advanced* tab.
4. In the *System Overhead Time Slice* text box, type or select the percentage of overhead time (10 -90%).
5. Click *OK*.

T

tag

A named area of the controller's memory where data is stored.

- Tags are the basic mechanism for allocating memory, referencing data from logic, and monitoring data.
- The minimum memory allocation for a tag is four bytes.
  - When you create a tag that stores a BOOL, SINT, or INT (which are smaller than four bytes), the controller allocates four bytes, but the data only fills the part it needs.
  - User-defined data types and arrays store data in contiguous memory and pack smaller data types into 32-bit words.

  The following examples show memory allocation for various tags:

  - *start*, which uses the BOOL data type:

Memory allocation	**Bits**	
	31 **1**	**0**
allocation	not used	*start*

  - *station_status*, which uses the DINT data type:

Memory allocation:	**Bits**
	31 **0**
allocation	*station_status*

  - *mixer*, which uses a user-defined data type:

Memory allocation	**Bits**			
	31 **24**	**23** **16**	**15** **8**	**7** **0**
allocation 1	*mixer.pressure*			
allocation 2	*mixer.temp*			
allocation 3	*mixer.agitate_time*			
allocation 4	unused	unused	unused	bit 0 *mixer.inlet* bit 1 *mixer.drain* bit 2 *mixer.agitate*

– *temp_buffer*, which is an array of four INTS (INT[4]):

Memory allocation:	Bits	
	31 16	0
allocation 1	*temp_buffer*[1]	*temp_buffer*[0]
allocation 2	*temp_buffer*[3]	*temp_buffer*[2]

See *alias tag*, *base tag*, *consumed tag*.

task

A scheduling mechanism for executing a program.

- By default, each new project file contains a pre-configured continuous task.
- You configure additional, periodic tasks, as needed.
- A task provides scheduling and priority information for a set of one or more programs that execute based on specific criteria.
- Once a task is triggered (activated), all the programs assigned (scheduled) to the task execute in the order in which they are displayed in the controller organizer.
- You can only assign a program to one task at a time.

See *continuous task*, *periodic task*.

timestamp

A ControlLogix process that records a change in input data with a relative time reference of when that change occurred.

U

uncached connection

With the MSG instruction, an uncached connection instructs the controller to close the connection upon completion of the mSG instruction. Clearing the connection leaves it available for other controller uses. See *connection, cached connection*.

unidirectional connection

A connection in which data flows in only one direction: from the originator to the receiver. See *connection, bidirectional connection*.

upload

The process of transferring the contents of the controller into a project file on the workstation.

If you do not have the project file for a controller, you can upload from the controller and create a project file. However, not everything that is stored in a project file is available from the controller. If you upload from a controller, the new project file will not contain:

- rung comments
- descriptions for tags, tasks, programs, routines, modules, or user-defined structures
- chains of aliases (aliases pointing to other aliases)

  Alias chains are not completely reconstructed from the controller. If there are several possible names for a data item, the firmware and software choose a best-fit alias that may not reflect how the alias was specified in the original project.

See *download*.

user-defined data type

You can also create your own **structures**, called a user-defined data type (also commonly referred to as a user-defined structure). A user-defined data type groups different types of data into a single named entity.

- Within a user-defined data type, you define the **members**.
- Like tags, members have a name and data type.
- You can include arrays and structures.
- Once you create a user-defined data type, you can create one or more tags using that data type.
- Minimize your use of the following data type because they typically increase the memory requirements and execution time of your logic:
  - **INT**
  - **SINT**

  For example, some system values use the SINT or INT data type. If you create a user-defined data type to store those values, then use the corresponding SINT or INT data type.

- If you include members that represent I/O devices, you must use ladder logic to copy the data between the members in the structure and the corresponding I/O tags. Refer to "Buffer I/O" on page 8-1.
- When you use the BOOL, SINT, or INT data types, place members that use the same data type in sequence:

more efficient

BOOL
BOOL
BOOL
DINT
DINT

less efficient

BOOL
DINT
BOOL
DINT
BOOL

- You can use single dimension arrays.
- You can create, edit, and delete user-defined data types only when programming offline.
- If you modify a user-defined data type and change its size, the existing values of any tags that use the data type are set to zero (0).
- To copy data to a structure, use the COP instruction. See the *Logix5000 Controllers General Instruction Set Reference Manual*, publication 1756-RM003.

See *structure*.

W

watchdog

Specifies how long a task can run before triggering a major fault.

- Each task has a watchdog timer that monitors the execution of the task.
- A watchdog time can range from 1 ms to 2,000,000 ms (2000 seconds). The default is 500 ms.
- The watchdog timer begins to time when the task is initiated and stops when all the programs within the task have executed.
- If the task takes longer than the watchdog time, a major fault occurs: (The time includes interruptions by other tasks.)
- A watchdog time-out fault (major fault) also occurs if a task triggered again while it is executing (periodic task overlap). This can happen if a lower-priority task is interrupted by a higher-priority task, delaying completion of the lower-priority task.
- You can use the controller fault handler to clear a watchdog fault. If the same watchdog fault occurs a second time during the same logic scan, the controller enters faulted mode, regardless of whether the controller fault handler clears the watchdog fault.

ATTENTION

If the watchdog timer reaches a configurable preset, a major fault occurs. Depending on the controller fault handler, the controller might shut down.

To change the watchdog time of a task:

1. Open the RSLogix 5000 project.
2. In the controller organizer, right-click `name_of_task` and select *Properties*.
3. Click the *Configuration* tab.
4. In the *Watchdog* text box, type a watchdog time.
5. Click *OK*.

A

address
assign indirect 7-1
enter 4-7
alias
create 6-3
use 6-1
array
create 3-10
index through 7-1
organize 3-1
produce large array 11-1
ASCII
build string 13-12
compare characters 13-4
configure serial port 12-3
configure user protocol 12-5
connect device 12-2
convert characters 13-6
decode message 13-8
enter characters 12-21
extract characters 13-2
manipulate characters 13-1
organize data 12-8
read characters 12-9
write characters 12-14

B

bar code
extract characters 13-2
test characters 13-4
branch
enter 4-3
buffer
I/O 8-1

C

chassis size 1-3
clear
major fault 9-4, 15-1
minor fault 17-1
codes
major fault A-1
minor fault A-3
communicate
other controllers 10-1
compare
ASCII characters 13-4
compliance tables B-5
configure
driver 9-1
load from nonvolatile memory 19-1
serial port for ASCII 12-3
user protocol for ASCII 12-5
consume
integers from PLC-5C 10-9
tag 10-1
controller
change properties 1-3
download 9-2
mode 9-3
shut down 16-1
suspend 16-1
verify 2-5
ControlNet
configure driver 9-1
produce and consume data 10-1
convert
ASCII characters 13-6
create
alias 6-3
array 3-10
driver 9-1
project file 1-1
routine 2-3
string 13-12
string data type 12-8
structure 3-8
subroutine 2-3
tag 3-10, 4-7
tag using Excel 3-11
user-defined data type 3-8

D

data
ASCII 12-8
definitions B-2
enter ASCII characters 12-21
force 14-1
produce and consume 10-1
data table 3-1
download 9-2
driver
configure 9-1

E

enter
address 4-7
ASCII characters 12-21
force 14-2
function block instruction 4-4
ladder instruction 4-3
Ethernet
configure driver 9-1
produce and consume tags 10-1
extract
ASCII characters 13-2

F

fault
clear 9-4, 15-1
create user-defined 16-1
develop routine to clear fault 15-1, 18-1
major fault codes A-1
minor fault codes A-3
monitor minor 17-1
test a fault routine 15-4
force
disable 14-6
enable 14-5
enter 14-2
LED 14-7
monitor 14-7
remove 14-6
tag 14-1
function block
controllers that support function blocks 2-1
enter 4-4

I

I/O
buffer 8-1
synchronize with logic 8-1
ICON 4-4
IEC 1131-3 compliance
data definitions B-2
instruction set B-4
introduction B-1
operating system B-2
program portability B-4
programming language B-3
tables B-5
indirect address 7-1
instruction
enter function block 4-4
enter ladder 4-3
instruction set B-4
IREF 4-4

L

LED
force 14-7
load a project 19-1
logic
enter function block instruction 4-4
enter ladder instruction 4-3

M

major fault
codes A-1
create user-defined 16-1
develop fault routine 15-1, 18-1
manipulate string 13-1
message
decode string 13-8
organize tags 10-11
minor fault
clear 17-1
codes A-3
logic 17-1
mode
controller 9-3
monitor forces 14-7

N

nonvolatile memory 19-1

O

OCON 4-4
open
routine 4-1
operating system B-2
OREF 4-4
organize
array 3-1
strings 12-8
structure 3-1
tag 3-1
tasks 2-2

P

PLC-5C
share data 10-6, 10-7, 10-9
produce
large array 11-1
tag 10-1
tags for PLC-5C 10-6, 10-7
program
portability B-4
program mode 9-3
programming language B-3
project
download 9-2
load from nonvolatile memory 19-1
store in nonvolatile memory 19-1
upload 9-4
project file
create 1-1

R

read
ASCII characters 12-9
routine
create 2-3
enter function block instructions 4-4
enter ladder instructions 4-3
languages 2-1
open 4-1
verify 4-10
RSLinx
configure 9-1
run mode 9-3
rung
enter 4-3

S

save 1-2
see also store a project
save as 1-2
send
ASCII characters 12-14
serial
cable wiring 12-2
configure port for ASCII 12-3
connect an ASCII device 12-2
shut down the controller 16-1
slot number 1-3
status
monitor 5-1, 5-2
store a project 19-1
string
compare characters 13-4
convert characters 13-6
create 13-12
data type 12-8
enter characters 12-21
extract characters 13-2
manipulate 13-1
organize data 12-8
read characters 12-9
write characters 12-14
string data type
create 12-8
structure
create 3-8
organize 3-1
subroutine
create 2-3
suspend
controller 16-1
symbol
create 6-1
system data
access 5-2

T

tag
assign 4-7
create 3-10, 4-7
create alias 6-3
create using Excel 3-11
enter 4-7
force 14-1
organize 3-1
organize for message 10-11
produce and consume 10-1
produce large array 11-1
share with PLC-5C 10-6, 10-7, 10-9
string 12-8
task
organize 2-2
test a fault routine 15-4
test mode 9-3

U

upload 9-4

user protocol
configure for ASCII 12-5
user-defined data type
create 3-8

V

verify
controller 2-5
routine 4-10

W

weight
convert 13-6
write
ASCII characters 12-14

Allen-Bradley
Publication Problem Report

If you find a problem with our documentation, please complete and return this form.

Pub. Title/Type Logix5000™ Controllers Common Procedures Programming Manual

Cat. No. 1756-Lx, 1769-Lx, 1789-Lx, 1794-Lx Pub. No. 1756-PM001C-EN-P Pub. Date June 2001 Part No. 957464-72

Check Problem(s) Type:	Describe Problem(s)			Internal Use Only
Technical Accuracy	text		illustration	
Completeness	procedure/step	illustration	definition	info in manual
What information is missing?	example	guideline	feature	(accessibility)
	explanation	other		info not in manual
Clarity				
What is unclear?				
Sequence				
What is not in the right order?				
Other Comments				
Use back for more comments.				

Your Name _____ Location/Phone _____

Return to: Marketing Communications, Allen-Bradley., 1 Allen-Bradley Drive, Mayfield Hts., OH 44124-6118Phone:(440) 646-3176
FAX:(440) 646-4320

PLEASE FASTEN HERE (DO NOT STAPLE)

Other Comments

PLEASE FOLD HERE

PLEASE REMOVE

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST-CLASS MAIL PERMIT NO. 18235 CLEVELAND OH

POSTAGE WILL BE PAID BY THE ADDRESSEE

Rockwell Automation

1 ALLEN-BRADLEY DR
MAYFIELD HEIGHTS OH 44124-9705

ASCII Character Codes

Character	Dec	Hex	Character	Dec	Hex	Character	Dec	Hex	Character	Dec	Hex
[ctrl-@] NUL	0	$00	SPACE	32	$20	@	64	$40	‘	96	$60
[ctrl-A] SOH	1	$01	!	33	$21	A	65	$41	a	97	$61
[ctrl-B] STX	2	$02	“	34	$22	B	66	$42	b	98	$62
[ctrl-C] ETX	3	$03	#	35	$23	C	67	$43	c	99	$63
[ctrl-D] EOT	4	$04	$	36	$24	D	68	$44	d	100	$64
[ctrl-E] ENQ	5	$05	%	37	$25	E	69	$45	e	101	$65
[ctrl-F] ACK	6	$06	&	38	$26	F	70	$46	f	102	$66
[ctrl-G] BEL	7	$07	‘	39	$27	G	71	$47	g	103	$67
[ctrl-H] BS	8	$08	(	40	$28	H	72	$48	h	104	$68
[ctrl-I] HT	9	$09	)	41	$29	I	73	$49	i	105	$69
[ctrl-J] LF	10	$l ($0A)	*	42	$2A	J	74	$4A	j	106	$6A
[ctrl-K] VT	11	$0B	+	43	$2B	K	75	$4B	k	107	$6B
[ctrl-L] FF	12	$0C	,	44	$2C	L	76	$4C	l	108	$6C
[ctrl-M] CR	13	$r ($0D)	-	45	$2D	M	77	$4D	m	109	$6D
[ctrl-N] SO	14	$0E	.	46	$2E	N	78	$4E	n	110	$6E
[ctrl-O] SI	15	$0F	/	47	$2F	O	79	$4F	o	111	$6F
[ctrl-P] DLE	16	$10	0	48	$30	P	80	$50	p	112	$70
[ctrl-Q] DC1	17	$11	1	49	$31	Q	81	$51	q	113	$71
[ctrl-R] DC2	18	$12	2	50	$32	R	82	$52	r	114	$72
[ctrl-S] DC3	19	$13	3	51	$33	S	83	$53	s	115	$73
[ctrl-T] DC4	20	$14	4	52	$34	T	84	$54	t	116	$74
[ctrl-U] NAK	21	$15	5	53	$35	U	85	$55	u	117	$75
[ctrl-V] SYN	22	$16	6	54	$36	V	86	$56	v	118	$76
[ctrl-W] ETB	23	$17	7	55	$37	W	87	$57	w	119	$77
[ctrl-X] CAN	24	$18	8	56	$38	X	88	$58	x	120	$78
[ctrl-Y] EM	25	$19	9	57	$39	Y	89	$59	y	121	$79
[ctrl-Z] SUB	26	$1A	:	58	$3A	Z	90	$5A	z	122	$7A
ctrl-[ESC	27	$1B	;	59	$3B	[	91	$5B	{	123	$7B
[ctrl-\] FS	28	$1C	<	60	$3C	\	92	$5C	|	124	$7C
ctrl-] GS	29	$1D	=	61	$3D	]	93	$5D	}	125	$7D
[ctrl-^] RS	30	$1E	>	62	$3E	^	94	$5E	~	126	$7E
[ctrl-_] US	31	$1F	?	63	$3F	_	95	$5F	DEL	127	$7F

Reach us now at www.rockwellautomation.com

Wherever you need us, Rockwell Automation brings together leading brands in industrial automation including Allen-Bradley controls, Reliance Electric power transmission products, Dodge mechanical power transmission components, and Rockwell Software. Rockwell Automation's unique, flexible approach to helping customers achieve a competitive advantage is supported by thousands of authorized partners, distributors and system integrators around the world.

Americas Headquarters, 1201 South Second Street, Milwaukee, WI 53204, USA, Tel: (1) 414 382-2000, Fax: (1) 414 382-4444
European Headquarters SA/NV, avenue Herrmann Debroux, 46, 1160 Brussels, Belgium, Tel: (32) 2 663 06 00, Fax: (32) 2 663 06 40
Asia Pacific Headquarters, 27/F Citicorp Centre, 18 Whitfield Road, Causeway Bay, Hong Kong, Tel: (852) 2887 4788, Fax: (852) 2508 1846

Publication 1756-PM001C-EN-P - June 2001
PN 957464-72
Supersedes Publication 1756-PM001B-EN-P - November 2000

© 2001 Rockwell International Corporation. Printed in the U.S.A.